TECHNISCHE UNIVERSITÄT DARMSTADT (D 17)
Fachgebiet Entwicklung von Anwendungssystemen
Fachbereich Rechts- und Wirtschaftswissenschaften



Dissertation
zur Erlangung des akademischen Grades
Doctor rerum politicarum
(Dr. rer. pol.)
Doktor der Wirtschafts- und Sozialwissenschaften

# Composite Application Systems

*Systematisches Konstruieren von Verbundanwendungen*
*unter Verwendung von BPMN*

vorgelegt von:
Dipl. Inf. Univ. Volker Stiehl aus Korbach

| | |
|---|---|
| Erstgutachter: | Prof. Dr. Erich ORTNER |
| Zweitgutachter: | Prof. Em. Dr. Dr.-Ing. E.h. Hartmut WEDEKIND |
| | Prof. Dr. Mathias WESKE |

| | |
|---|---|
| Erscheinungsort/-jahr: | Darmstadt, 2011 |
| Einreichungsdatum: | 07. Juli 2011 |
| Prüfungsdatum: | 05. September 2011 |

## Zusammenfassung

Unternehmen stehen in der heutigen globalisierten Welt, in der Veränderungen und Wandel ihr ständiger Begleiter sind, vor enormen Herausforderungen: sie müssen sich innerhalb kürzester Zeit diesem permanenten Wechsel anpassen, der mit neuen technischen Weiterentwicklungen einhergeht, da sie ansonsten ihre Existenz gefährden. Idealerweise kann die Unternehmens-IT bei der Bewältigung dieser Herausforderungen behilflich sein, indem sie ihren Beitrag zur schnellstmöglichen Umsetzung der Unternehmensstrategie leistet. Allerdings bleibt in vielen Fällen die Kernfrage unbeantwortet: *wie* kann die Unternehmens-IT konkret eine beschleunigte Implementierung neuer strategischer und dabei gleichzeitig differenzierender Prozesse gewährleisten, damit sie zu einer tragenden Stütze der Unternehmensleitung wird?

Die Überlebenschancen von Unternehmen vergrößern sich, wenn sie sich auf Geschäftsprozesse konzentrieren, die ihren Kunden einen deutlichen Mehrwert liefern und die sich dabei gleichzeitig von der Konkurrenz abheben. Derartige Prozesse finden sich in keiner Standardsoftware, sie müssen individuell entworfen und implementiert werden. In dieser Arbeit wird eine besondere Klasse von Anwendungssystemen mit einer speziellen Architektur betrachtet, die eine Balance zwischen den Anforderungen nach Flexibilität, Nachhaltigkeit, Wartbarkeit bei gleichzeitiger moderater Komplexität findet und somit genau das Potenzial besitzt, sich als die gesuchte Stütze zu erweisen: die der Verbundanwendung (Composite Application).

Als zentrales Ergebnis dieser Arbeit wird ein Entwurfs- und Implementierungsansatz vorgeschlagen, der durch die fachlichen Geschäftsprozesse getrieben wird und sich auf die Kombination der folgenden drei Säulen stützt:

1. Der Einsatz einer konsequent top-down-orientierten Methodologie zur Bestimmung der wesentlichen Bestandteile/Artefakte einer Verbundanwendung.
2. Einer nachhaltigen Architektur für Verbundanwendungen, die zwischen den eigentlichen Fachprozessen auf Verbundanwendungsebene und den technischen Prozessen in der sogenannten Servicevertrag-Implementierungsschicht unterscheidet. Dabei wird gleichzeitig sowohl eine eindeutige Auf-

gabenzuordnung zu den beiden Schichten als auch deren Zusammenspiel festgelegt. Die Trennung bewirkt einen umfassenden Schutz der fachlichen Prozesse gegenüber den ständigen Veränderungen auf Systemebene.

3. Der durchgängigen Verwendung der Business Process Model and Notation (BPMN) sowohl zur Modellierung als auch zur Implementierung sämtlicher Prozessanteile auf Verbundanwendungsebene und der Ebene der Servicevertrag-Implementierungsschicht. In der Servicevertrag-Implementierungsschicht kann zudem zwischen zustandsbehafteten und zustandslosen Prozessen unterschieden werden.

Nachdem zu Beginn der Arbeit ein Verständnis für die Eigenschaften, die Architektur sowie die Alleinstellungsmerkmale von Verbundanwendungen aufgebaut wird, geht es im weiteren Verlauf um deren Implementierung. Es wird der Frage nachgegangen, wie eine Umsetzung von Verbundanwendungen schnellstmöglich gewährleistet werden kann. Ein besonderer Fokus wird hierbei auf die Eignung der BPMN für diese Aufgabe gelegt und herausgearbeitet, wie BPMN bei der Implementierung der verschiedenen Schichten eingesetzt werden kann. Ob eine Verbundanwendung nun einer Zwei-Schichten- oder Drei-Schichten-Architektur folgt – BPMN zeichnet sich allein durch die bereits im Standard beinhalteten Funktionalitäten aus, die eine elegante Realisierung lose gekoppelter Architekturen, wie sie bei Verbundanwendungen üblich sind, erlaubt. Es wird gezeigt, wie sich BPMN insbesondere im Bereich der Systemintegration bewährt, da Verbundanwendungen darauf ausgelegt sind, soweit möglich existierende Funktionalitäten aus der bestehenden IT-Landschaft wiederzuverwenden und Integration dadurch zu einem weiteren zentralen Thema dieser Arbeit werden lässt. Dazu werden verschiedene BPMN-basierte Integrationspattern entwickelt: sowohl aktive Pattern, die den Nachrichtenverkehr über Request/Response treiben als auch passive Pattern, die auf eintreffende Nachrichten warten und die Nachrichtensequenzen koordinieren. Aufgrund der gewonnenen Erkenntnisse wird schließlich eine Erweiterung der BPMN vorgeschlagen, um die Vorteile von BPMN mit einer bei Systemintegratoren etablierten Notation, die auf den von Gregor Hohpe und Bobby Woolf beschriebenen Enterprise Integration Patterns basieren, zu verbinden und um so zu einem noch mächtigeren Werkzeug zur Darstellung von Integrationsprozessen zu gelangen.

## Abstract

In a globalized world enterprises are facing tough challenges as changes are permanently accompanying them: as a consequence companies have to adapt their businesses in even shorter timeframes to this ever changing world which is caused by constant improvements in technology. If they do not follow these changes they are threatening their existence. Ideally the company's IT department can help in overcoming those challenges by taking their part in implementing the company's strategy as fast as possible. However, very often the key question stays unanswered: *how* can an IT department ensure the fast implementation of new strategic and differentiating processes so that they become a key pillar for a company?

The chances of survival for companies increase if they concentrate on business processes which deliver a significant value to their customers and at the same time differentiate them from their competition. These kinds of processes cannot be found in standard software, they have to be planned and developed individually. In this thesis a special class of application systems with a particular architecture will be analyzed which has the potential to find the right balance between requirements such as flexibility, sustainability, maintainability, and a moderate complexity and with that has the chance to be exactly the pillar companies are searching for: composite applications.

As key result of this thesis a design- and implementation approach is proposed which is driven by the business processes and relies on the combination of the following three pillars:

1. The adoption of a totally top-down-oriented methodology for deriving the essential parts/artifacts of a composite application.
2. A sustainable architecture for composite applications, which differentiates between the business processes on the level of the composite application and the technical processes within the so-called service contract implementation layer. At the same time the tasks of each layer will be defined as well as their cooperation. This separation protects the business processes from the constant changes on system level.

3. The general usage of the Business Process Model and Notation (BPMN) for modeling as well as implementing of all processes within the composite application itself and within the service contract implementation layer. In addition on the level of the service contract implementation layer processes can be further differentiated between stateful and stateless processes.

After gaining an understanding about the key characteristics of composite applications (such as architecture and unique value propositions), their concrete implementation will be discussed next. The question will be answered, how a fast implementation can be achieved. The thesis will primarily focus and discuss the usage of BPMN (Business Process Model and Notation) for this purpose and will work out, how BPMN can be used on different layers of a composite application. Whether a composite application is following a two-tier or three-tier architecture – BPMN can be used in any of those approaches due to functions and features of the notation which are already part of the BPMN standard itself and which allow an elegant implementation of loosely coupled architectures. It will be shown how BPMN is especially suited for system integration processes as composite applications are heavily relying on reusing existing functionality residing in the available IT landscape. As a natural consequence integration becomes another major topic of this thesis. Therefore BPMN-based integration patterns will be developed which support both kinds of interactions: integration processes which actively drive the communication via request/response as well as processes passively waiting for messages and coordinating only the message sequence. Based on the made observations finally an extension of BPMN will be proposed, which combines the advantages of BPMN with a symbol notation first recommended by Gregor Hohpe and Bobby Woolf for enterprise integration patterns. This combination results in a more powerful notation especially suited for depicting integration processes.

# Inhalt

# 1 Einleitung

Heutige Unternehmen sehen sich aufgrund der Globalisierung einem erhöhten Wettbewerb ausgesetzt. Sie müssen im alltäglichen Kampf ums Überleben schneller und nachhaltiger als die Konkurrenz auf Veränderungen reagieren können. Besser als das folgende von Darwin stammende Zitat kann man die derzeitige Situation von Unternehmen wohl kaum beschreiben:

> It is not the strongest of the species that survives,
> nor the most intelligent that survives.
> It is the one that is the most adaptable to change.

Es ist genau diese Fähigkeit von Unternehmen, auf Veränderungen adäquate Antworten parat zu haben, die ihnen ihre Existenz letztendlich garantieren. Dabei kann die Informationstechnologie maßgeblich beitragen: Dank enormer Fortschritte im Software-Engineering, der modellgetriebenen Softwareentwicklung, Standardisierung von Kommunikations- und Schnittstellentechnologien über Web Services, der Modularisierung betriebswirtschaftlicher Funktionalitäten in Form von wiederverwendbaren Services sowie der Produktivitätssteigerungen für Software-Entwickler aufgrund hochintegrierter Entwicklungsumgebungen begleitet durch agile Entwicklungsmethoden ist die effiziente Bereitstellung neuer Software-Lösungen weniger das Problem. Stattdessen erweisen sich aus Gesamtkostenbetrachtung, der Total Cost of Ownership (TCO) also, die Pflege, Erweiterung sowie Anpassung der Software an veränderte Bedingungen als viel größere Hürden: aufgrund erhöhten Kostendrucks entstehen Softwarearchitekturen, die genau der immer wichtiger werdenden Fähigkeit eines Softwaresystems zur Anpassbarkeit entgegenwirken. Dies gilt insbesondere für Software, die bereits existierende betriebswirtschaftliche Logik zu neuen, system- und anwendungsübergreifenden Applikationen gemäß der SOA-Idee zusammenfügen soll. Der ursprüngliche Gedanke einer serviceorientierten Architektur, nämlich Potenziale wie gesteigerter Agilität, geringere Kosten, sowie eine zeitnahe Umsetzung der Unternehmensstrategie in Software heben zu wollen, sind in ersten Projekten nicht erfüllt worden (Manes 2009). Gründe für dieses Scheitern sind unter anderem in der Übernahme der aus der Objektorientierung stammenden Erfahrungen in die serviceorientierte Welt

zu finden: Architekturen, die in eng-gekoppelten Standalone-Anwendungen vollkommen richtig waren, sind in lose-gekoppelten verteilten Anwendungen genau der verkehrte Ansatz. Ob es um die Verwendung von Datentypen, der Kommunikation mit Systemen oder das Transaktionsverhalten geht – in einer SOA-Umgebung sind dazu andere Lösungen notwendig. Sie verlangen ein Umdenken bei den Entwicklern: so wie bei jedem Paradigmenwechsel sind veränderte Herangehensweisen die logische Konsequenz. Ob beim Wechsel von der maschinennahen zur strukturierten Programmierung, ob von der strukturierten zur objektorientierten Programmierung oder eben wie zur Zeit der Wandel hin zur Entwicklung von serviceorientierten Architekturen: in jeden dieser Evolutionsschritte nimmt die Abstraktion zu und verlangt eine Auseinandersetzung des Entwicklers mit den neuen Paradigmen, um optimal von den neuen Möglichkeiten profitieren zu können. Alte Gewohnheiten in die neue Welt zu transportieren hilft in den wenigsten Fällen und ist daher eher hinderlich. Insbesondere der Aspekt, dass man bei der Entwicklung derartiger webbasierter Verbundanwendungen oder Composite Applications – beide Begriffe werden im Laufe dieser Arbeit synonym verwendet – eben nicht mehr die alleinige Kontrolle über die beteiligten Ressourcen, wie zum Beispiel einem Datenbanksystem, besitzt und man sich zudem in einer heterogenen Systemlandschaft bewegt, in der es existierende Dienste zu neuer Geschäftslogik zu kombinieren gilt, erfordert ein gesteigertes Problembewusstsein bei den Anwendungsentwicklern.

Damit sind die Anforderungen an derartige Systeme aber bei weitem nicht erschöpft. Der Flexibilitätsaspekt ist bei solchen Anwendungen ebenfalls nicht zu unterschätzen. Er lässt sich konkret an zwei Anforderungen verdeutlichen:

1. Änderungen innerhalb der Verbundanwendung selbst, insbesondere Änderungen der in der Anwendung abgedeckten Geschäftsprozesse
2. Änderungen innerhalb der Systemlandschaft, auf der die Verbundanwendung basiert

Geschäftsprozesse (im klassischen Sinne zu verstehen als „eine Reihe von festgelegten Tätigkeiten (Aktivitäten, Aufgaben), die von Menschen oder Maschinen ausgeführt werden, um ein oder mehrere Ziele zu erreichen“ aus EABPM 2009, siehe auch Hammer und Champy 1993 oder Gaitanides 1983) spielen bei Verbundanwendungen

eine besondere Rolle: um sich von der Konkurrenz absetzen zu können, müssen Unternehmen ständig an den Geschäftsprozessen feilen, die ihnen einen Wettbewerbsvorteil sichern. Erfolgreiche Prozesse werden dabei unweigerlich von der Konkurrenz kopiert, so dass der Vorteil wieder schrumpfen wird. Hier sei beispielhaft an Flugunternehmen erinnert: die Bereitstellung von Selbstbedienungsterminals zum Einchecken brachte dem Unternehmen, das diese Idee als Erstes umsetzte, seinerzeit einen entscheidenden Vorteil und damit mehr Kunden. Heute stellen nahezu alle Fluglinien einen solchen Service zur Verfügung, so dass der Vorteil wieder verloren ging. Auch geänderte Marktbedingungen verlangen von den Unternehmen eine schnellstmögliche Anpassung ihrer Prozesse. Was auch immer der Auslöser sein mag, die Folgen werden stets dieselben sein: ein neuer Innovationszyklus zwingt Unternehmen schließlich dazu, weitere Optimierungen an den geschäftskritischen Prozessen vorzunehmen, um weiterhin erfolgreich agieren zu können. Je schneller solche Änderungen implementiert werden können, umso wahrscheinlicher wird das Vorgehen von Erfolg gekrönt sein. Folglich muss die Änderungsfreundlichkeit der Software in der Architektur von Verbundanwendungen bereits verankert sein, um den Bedarf nach Flexibilität und Agilität gerecht werden zu können.

Auf der anderen Seite müssen Verbundanwendungen auf eine sich ständig ändernde Systemlandschaft vorbereitet sein. Es gehört zum Kern von Composite Applications, dass sie in einem heterogenen Systemumfeld agieren: der Wiederverwendungsgedanke ist fundamental für SOA-basierte Anwendungen. Das Rad soll eben nicht neu erfunden werden und wo etablierte, funktionierende Geschäftslogik bereits existiert und über Standardkommunikationswege wie Web Services abgreifbar ist, so soll diese auch entsprechend genutzt werden. Allerdings ergibt sich daraus eine besondere Herausforderung: derartige Systemlandschaften sind nicht stabil. Auch hier müssen Veränderungen von vornherein bei der Konzeption von Verbundanwendungen berücksichtigt werden. Mindestens drei Gründe sprechen dafür, dass sich Systemlandschaften auch in Zukunft ständig, vielleicht sogar in einem noch höheren Tempo, verändern werden:

1. Systemkonsolidierungen
2. Firmenfusionen
3. Software as a Service (SaaS)

IT-Abteilungen in Unternehmen unterliegen einem enormen Kostendruck. Die Unternehmensleitung verlangt von IT-Verantwortlichen ständige Kostenoptimierung. System-Konsolidierungen sind da ein probates Mittel: je mehr Systeme und Funktionalitäten zusammengelegt werden können, umso geringer werden die Belastungen. Allein aus diesem Grund werden Systemlandschaften sich ständig weiterentwickeln. Für Verbundanwendungen bedeutet dies, dass Aufrufe, die zuvor gegen mehrere Systeme erfolgten, nun gegen eine reduzierte Anzahl von Systemen abgewickelt werden müssen.

Doch das ist nur die eine Seite der Medaille: durch das Abschalten von Systemen und der Überführung von ehemals differierenden Prozessen in Standardsoftware wird zwar eine Vereinfachung der IT-Landschaft erreicht. Diesem Trend entgegen wirkt jedoch die Fusion von Unternehmen. Im Zuge von Wachstumszielen für die Unternehmung, wird das organische Wachstum durch Firmenakquisitionen unterstützt. Allerdings wächst durch derartige Firmenzusammenschlüsse naturgemäß der System- und Anwendungspark. In diesen Fällen ist die Composite Application gemäß der geänderten Rahmenbedingungen an die gestiegen Systemanzahl anzupassen.

Schließlich muss man dem neuen Trend der Software-as-a-Service (SaaS)-Anbieter Rechnung tragen: betriebswirtschaftliche Dienste werden jetzt kostengünstig von Dienstleistern angeboten mit weiterem Optimierungspotenzial für IT-Leiter. Sie können jetzt unkritische Funktionalitäten auslagern und somit zu einer weiter optimierten Kostenstruktur für die IT beitragen. Für die Architektur von Verbundanwendungen bedeutet allerdings auch dieses Szenario einen erhöhten Bedarf nach Flexibilität. Welches Szenario auch immer zum Tragen kommt, ob Systemkonsolidierungen, Hinzufügen neuer Systeme oder die Verlagerung von Funktionalitäten: für Composite Applications gehören derartige Herausforderungen zum Alltag und müssen bei deren Konzeption integraler Bestandteil sein.

Auch wenn die bisherigen Ausführungen den Eindruck erwecken lassen, dass Verbundanwendungen nur für größere Unternehmen von Bedeutung sind, so täuscht dies. Sicherlich spielt der Aspekt einer sich ständig ändernden Systemlandschaft für kleine und mittelständische Unternehmen (KMU) eine eher untergeordnete Rolle, obwohl SaaS sicherlich auch bei ihnen Spuren hinterlassen wird. Aber auch für sie gilt: nur erfolgreiche, gegenüber der Konkurrenz differenzierende Prozesse garantieren das

Überleben. Hier unterscheiden sich die Anforderungen im Vergleich zu Großunternehmen in keinster Weise und damit gilt auch für sie hinsichtlich der Bedeutung von Änderungsfreundlichkeit als Bestandteil der Applikationsarchitektur das oben Gesagte.

Doch nicht nur innerhalb von Unternehmen spielt die neue Generation von Verbundanwendungen ein besondere Rolle. Auch Softwarehersteller (ISVs – Independent Software Vendors) können auf Basis von Composite Applications attraktive neue Märkte erschließen. In der Regel ist das Geschäftsmodell für ISVs darauf ausgerichtet, Lösungen zu entwickeln, die als Ergänzungen zu den Produkten bekannter Hersteller wie beispielsweise SAP, Oracle oder Microsoft zu sehen sind und die sich auch nur gegen diese Standardlösungen verbinden können. Das Problem, das sich aufgrund dieses Modells ergibt, liegt auf der Hand: zeigt ein Kunde Interesse an der betriebswirtschaftlichen Lösung des Herstellers, so scheiterte das Zustandekommen eines Geschäfts ganz einfach an den fehlenden Voraussetzungen beim Kunden: die benötigte Standardanwendung ist dort nicht im Einsatz und ist auch kurzfristig nicht geplant. Viele potenzielle Kunden scheiden durch das Festlegen auf einen Hersteller von vornherein aus. Könnte also Software erstellt werden, die sich den anzutreffenden Systemlandschaften beim Kunden anpassen ließe, so würden sich gänzlich neue Perspektiven für ISVs ergeben. Und genau in diese Lücke stoßen Verbundanwendungen. Einmal erstellt, erlaubt deren Architektur die Anpassbarkeit an die beim Kunden angetroffene Landschaft, ohne dass die eigentliche betriebswirtschaftliche Lösung davon betroffen ist.

Die zuvor genannten Eigenschaften, Probleme und Szenarien stellen den Ausgangspunkt dieser Arbeit dar. Es wurde nach Aufkommen der SOA-Idee und des damit verbundenen Hypes eine Vielzahl von Büchern und Artikeln zum SOA-Thema verfasst, die einzelne Aspekte teilweise auch sehr tiefgehend diskutieren. Was allerdings fehlt ist die detaillierte und dabei ganzheitliche Beschreibung der Architektur von betriebswirtschaftlich-orientierten Verbundanwendungen sowie deren konkrete Implementierung, die das volle Potenzial einer serviceorientierten Architektur ausschöpft und dabei die SOA-Versprechen auch erfüllt. Die einzelnen SOA-relevanten Aspekte wie beispielsweise lose Kopplung, Transaktionshandling über Kompensation, Fehlertoleranz, Rolle von Service-Repositories, Anforderungen an Services, damit

diese eine serviceorientierte Architektur und damit Verbundanwendungen optimal unterstützen usw. müssen dafür zu einem harmonischen Ganzen zusammengefügt werden. Dabei ist diese Arbeit gleichzeitig ein Plädoyer für nachhaltige Architekturen, die sich im Alltag als robust bewähren und die rasche Anpassungen erlauben, und zwar so, wie die Geschäftsleitung im Idealfall ihr Geschäft abgewickelt sehen möchte. Die Zeitdifferenz zwischen der Vorgabe einer neuen (Geschäfts-) Strategie und deren Umsetzung wird Dank einer durchdachten Architektur auf ein Minimum reduziert.

Auch bei der bereits angesprochenen Implementierung werden im Rahmen dieser Arbeit neue Wege beschritten. Dabei soll die Verwendung und Eignung der Business Process Model and Notation (BPMN) in ihrer neuesten Version 2.0 für genau diesen Zweck analysiert werden, da gerade im Hinblick auf die Umsetzung sowohl kollaborativer aber auch systemzentrischer Prozesse interessante Neuerungen in den Standard eingeflossen sind. Diese neuen Möglichkeiten zu eruieren, in konkreten Beispielen anzuwenden, sowie Möglichkeiten und Grenzen auszuloten wird ebenfalls Gegenstand der Betrachtungen sein.

Damit diese Ziele erreicht werden können, wird es im weiteren Verlauf der Arbeit zunächst darum gehen, den Begriff der Verbundanwendung zu schärfen (Kapitel 2). Es geht um die typischen Eigenschaften derartiger Anwendungen und dabei gleichzeitig um die Abgrenzung zu anderen Anwendungstypen wie eng-gekoppelten Einzelplatzprogrammen, projektspezifischen Lösungen oder reinen Integrationsapplikationen. Einige Beispiele verdeutlichen die Ideen, die den Composite Applications zugrunde liegen.

Nachdem auf diese Weise ein Verständnis von Verbundanwendungen aufgebaut wurde, wird sich Kapitel 3 mit der grundlegenden Architektur von Composite Applications sowie der Methode beschäftigen, mit der die wesentlichen Bestandteile einer Composite wie Prozesse, Benutzeroberflächen, Daten in Form von Geschäftsobjekten sowie die benötigten Dienste ermittelt werden können. Kapitel 4 behandelt eine konkrete Implementierung einer Verbundanwendung. Eine wesentliche Rolle wird dabei, wie bereits erwähnt, die standardisierte grafische Notation zur Modellierung von Geschäftsprozessen BPMN spielen. Warum BPMN in diesem Zusammenhang so wichtig ist, wird Gegenstand einer ausführlichen Diskussion sein. Es wird aber nicht nur bei einer theoretischen Abhandlung bleiben. Es wird anhand eines kleinen Beispiels zu-

dem gezeigt, dass die zu erwartenden Effekte auch tatsächlich erzielt werden können. Dafür wird anhand eines Einkaufsprozesses die Umsetzung in Software unter Zuhilfenahme des SAP NetWeaver Composition Environments vorgestellt, die wesentlichen Implementierungsschritte kurz erklärt und abschließend die dabei gemachten Erfahrungen diskutiert.
Bei Composite Applications handelt es sich um komponentenbasierte Anwendungen, die selbst wiederum von der Wiederverwendung von Komponenten abhängig sind. Da liegt es natürlich nahe, aktuelle Forschungsergebnisse rund um die Modellierung der Aufbaustruktur von Composites auch unter Berücksichtigung von Komponenten-Varianten zu berücksichtigen. Eine entsprechende Behandlung dieses Themas findet sich ebenfalls in Kapitel 4.

Verbundanwendungen decken unternehmenskritische betriebswirtschaftliche Prozessabläufe ab und das trotz verteilter Architekturen und potenziell vielfältiger Fehlersituationen. Damit sie dabei dennoch eine ähnliche Robustheit und Zuverlässigkeit wie herkömmliche eng-gekoppelte Anwendungen erreichen, reicht eine solide Architektur nicht aus. Begleitende Konzepte wie optimistisches Locking in den beteiligten Systemen, Idempotenz-Unterstützung bei den Services, Transaktionsverarbeitung über Kompensation, das Konzept der Fehlerbehandlung nahe bei den involvierten Backend-Systemen usw. helfen bei der Stabilisierung der Gesamtlösung. Wie diese ergänzenden Maßnahmen dabei zusammenwirken, ist Gegenstand von Kapitel 5.

Ebenfalls in Kapitel 5 werden typische Prozessfragmente (Pattern) vorgestellt, die bei der Verbindung von Verbundanwendungen mit den Systemen der IT-Landschaften einsetzbar sind. In diesem Zusammenhang wird eine Erweiterung der BPMN vorgeschlagen, um insbesondere Integrationsprozesse noch prägnanter und präziser modellieren zu können. Fortgesetzt wird Kapitel 5 mit einer Diskussion unterschiedlichster Ansätze zur Flexibilitätssteigerung sowohl der Verbundanwendung selbst, als auch der Integration mit den Backend-Systemen. Regelwerke (Business Rules) und analytische Anwendungen werden hierbei eine wichtige Rolle übernehmen..

Abschließend werden im Ausblick des Kapitels 5 gegenwärtige Diskussionen rund um strukturierte und unstrukturierte Prozesse analysiert sowie mögliche Auswirkungen auf Verbundanwendungen erläutert.

Kapitel 6 fasst die Ergebnisse der Arbeit zusammen und behandelt zu guter Letzt das Thema, wie Composites als Produkt betrachtet durch Kunden modifikationsfrei

erweitert werden können. Hier zeichnen sich erste Ansätze ab, auf die kurz eingegangen werden soll.

An dieser Stelle muss betont werde, dass es nicht Gegenstand dieser Arbeit sein kann, alle Facetten der Entwicklung serviceorientierter Software zu beleuchten. Der Schwerpunkt liegt eindeutig auf der Ausarbeitung einer konkreten Architektur für betriebswirtschaftliche Anwendungen, die das Potenzial von SOA voll ausschöpfen, sowie deren Umsetzung auf Basis von BPMN. Natürlich können auch abweichend von der vorgeschlagenen Architektur SOA-basierte Anwendungen entwickelt werden, die nur Teile des vorgeschlagenen Ansatzes umsetzen. Architekten solcher Lösungen sind dann aber darüber informiert (und dazu möchte diese Arbeit auch einen Beitrag leisten), welche möglichen Nachteile sich bei Abweichungen von der vorgeschlagenen Lösung ergeben und können dementsprechend fundierte Entscheidungen treffen..

Themen, die sich hingegen weniger auf die Architektur einer Composite auswirken, wie beispielsweise…

- Organisatorische Aspekte einer SOA
- Lebenszyklus von Services
- Versionsmanagement
- SOA und Sicherheit
- Enterprise Service Bus
- SOA Governance
- SOA Projektmanagement
- Suchen und Finden von Services

sind bewusst ausgespart worden, da sie als Einzelthemen in der Literatur ausgiebig diskutiert wurden. Gute Einführungen zu diesen Themen finden sich u.a. in Krafzig et al., 2004 sowie Josuttis, 2008.

# 2 Definition von Verbundanwendungen

In diesem Kapitel geht es darum, das Verständnis über Verbundanwendungen zu konkretisieren, damit im darauffolgenden Kapitel eine passende Architektur erarbeitet werden kann. Die Idee der Verbundanwendung an sich ist nicht neu. Insbesondere die Firma SAP hat über viele Jahre hinweg das Bild von Composite Applications geprägt, weshalb an dieser Stelle ein kleiner geschichtlicher Exkurs die wesentlichen Meilensteine zusammenfasst.

Bereits im Jahre 2002 kündigte SAP im Rahmen ihrer Kundenkonferenz Sapphire eine neue Anwendungsgeneration namens "Cross Applications" (xApps) an, die auf den Prinzipien der Verbundanwendung fußt (SAP 2002a). In der Mitteilung wurden auch gleich vier wichtige Eigenschaften von Verbundanwendungen genannt, die nach wie vor ihre Gültigkeit besitzen und daher hier kurz zitiert werden sollen (SAP 2002a):

1. xApps are cross functional, connecting multiple applications from any vendor, even across company boundaries (Idee der funktions- und systemübergreifenden Applikation, die auch über Unternehmensgrenzen hinweg agiert).

2. These applications are composite, supporting end-to-end processes by adding functionality through components (Idee der Ende-zu-Ende-Prozesse und der Komponentisierung).

3. xApps are collaborative, unleashing the power of teams to drive change and innovation in a shared business context, and providing qualified information to enable collaborative decision making (Idee der benutzerzentrierten Kollaboration).

4. xApps leverage the mySAP™ Technology platform for the integration of people, information, and business processes, allowing the applications to support heterogeneous IT landscapes using industry standards (Idee der Verwendung von Industriestandards zur Kommunikation, insbesondere Web Services, und der Integration von Menschen, Informationen und Geschäftsprozessen).

Im selben Jahr wurden erste technische Details der Architektur während der für Entwickler bestimmten Konferenz SAP TechEd veröffentlicht (SAP 2002b). Im Jahre 2003 erschien dann das Buch *Packaged Composite Applications* von Dan Woods (Woods 2003), das in enger Zusammenarbeit mit der SAP entstanden ist und in dem ausführlich auf die Composite-Ideen eingegangen wird. Hervorzuheben sind die Ausarbeitung der Grundlagen der Verbundanwendung, deren Definition sowie ihrer charakteristischen Eigenschaften. Allerdings fehlt in all diesen frühen Veröffentlichungen ein detaillierter technischer Blueprint, anhand dessen sich sowohl SAP-interne Entwicklungsabteilungen als auch SAP-Partner hätten orientieren können, wie diese Ideen nun konkret umzusetzen sind.

Erst im Jahre 2005 wurde das Projekt iCOD (Industry Composite Development) initiiert, dessen Aufgabe es war, konkrete Vorschläge für den Bau von Verbundanwendungen unter Verwendung der SAP-Technologien, -Werkzeuge und -Frameworks auszuarbeiten. Die Ergebnisse wurden zunächst in mehreren SAP-internen Papieren (SAP 2005, SAP 2007 und SAP 2008) zusammengefasst und anschließend in überarbeiteter Form auch der Allgemeinheit zur Verfügung gestellt (Schuller 2007a, 2007b, 2007c). Zudem wurden die Ideen in den darauffolgenden Jahren während der diversen SAP TechEd-Konferenzen veröffentlicht. Diese Ideen bilden auch für die vorliegende Arbeit das Fundament. Wie noch zu sehen sein wird, ist insbesondere die Aufteilung der Composite-Funktionalität auf Prozess-, Benutzeroberflächen- und Geschäftslogik-Schicht identisch. Bei der Anbindung der zu integrierenden Backend-Systeme hingegen, also *dem* Kernthema bei der Verbundanwendungsentwicklung schlechthin, zeigen sich doch gravierende Unterschiede. So heißt es in der Empfehlung aus Schuller 2007a (S. 9) wörtlich:

As composite applications are defined as applications built on top of other applications which reuse existing functionality via service calls, the question is how the consumption of those services actually works. The recommended solution is the web service technology.

Diese Empfehlung kann heute so nicht mehr gegeben werden und lässt sich rückblickend wohl damit begründen, dass im Zuge der SOA-Bewegung im Allgemeinen die Einbeziehung externe Systeme über synchrone Mechanismen bevorzugt dargestellt wurde, um die Einfachheit der Integration zu veranschaulichen. Dabei wurde der enormen Bedeutung der Integration gerade in einer SOA, deren Einsatz in heterogenen Umgebungen prädestiniert ist, vielfach zu wenig Beachtung geschenkt. Hohpe & Woolf 2004 führen dies auf einen Integrationsansatz zurück, der versucht, Kommunikation mit externen Systemen auf dieselbe Semantik wie der bei lokalen Methodenaufrufen zu reduzieren (Hohpe & Woolf 2004, S. 10). Die Autoren führen dafür zwei naheliegende Gründe ins Felde:

1. Anwendungsentwicklern ist der Umgang mit lokalen Aufrufen sehr vertraut. Warum sollte man also nicht auf Bekanntem aufbauen?
2. Die Verwendung derselben Syntax sowohl für lokale als auch entfernte Methodenaufrufe erlaubt es, erst zum Deploy-Zeitpunkt darüber zu entscheiden, welche Methoden lokal und welche entfernt abzuwickeln sind.

Dabei weisen Waldo et. al. 1994 in ihrem Papier *A Note on Distributed Computing* bereits daraufhin, dass entfernte Aufrufe nicht mit lokalen vergleichbar sind und folglich auch unterschiedlich zu behandeln sind. Die naheliegende Schlussfolgerung fassen Hohpe & Woolf 2004 sehr treffend wie folgt zusammen (Hohpe & Woolf 2004 S. 11):

In summary, trying to portray remote communication as a variant of a local method invocation is asking for trouble. Such architectures typically result in brittle, hard to maintain and poorly scalable solutions. Many Web services pioneers recently (re-)discovered this fact the hard way.

Zwar wird auch in den ersten SAP-Papieren der Einsatz eines Message Brokers in Erwägung gezogen, doch bleibt eine detaillierte Beschreibung seiner Funktionsweise und der zu übernehmenden Aufgaben aus. Stattdessen wird zur Abstraktion der SAP-Backend-Systeme ein sogenannter Backend Abstraction Layer (BAL) vorgeschlagen (vgl. Abbildung 1), der Bestandteil der Verbundanwendung selbst ist, ebenfalls ein

Punkt, der im Laufe dieser Arbeit noch sehr intensiv diskutiert und zu einem anderen Ergebnis führen wird.

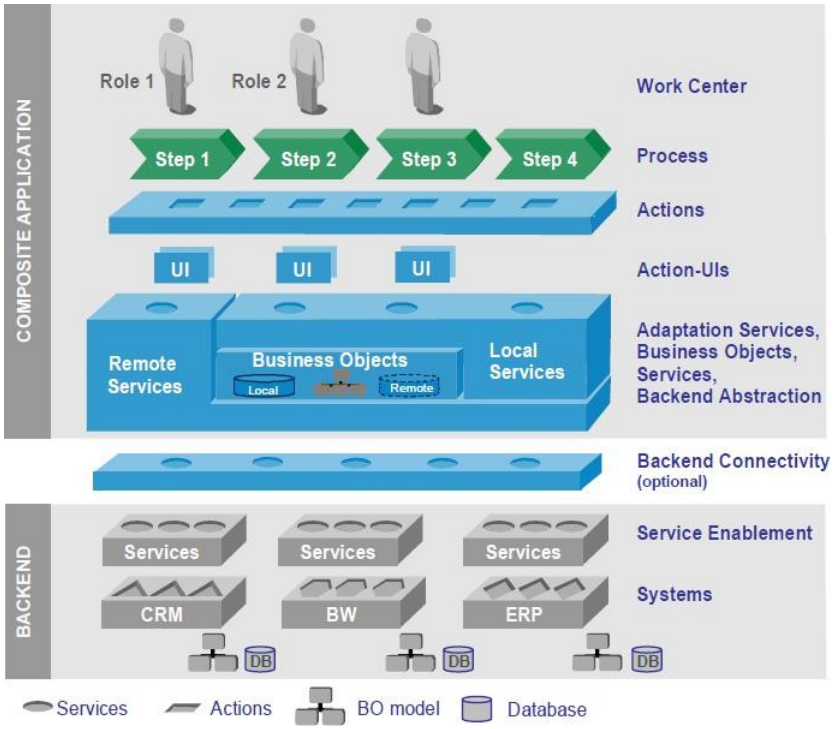



***Abbildung 1: Architektur einer Composite Application (aus Schuller 2007a, S. 8)***

Der wesentliche Gedanke des BAL, der programmatisch mittels Enterprise Java Beans (EJB)-Technologie umgesetzt werden sollte, liegt in der Abstraktion der verschiedenen von SAP zur Verfügung gestellten Schnittstellen zum Abgreifen von Logik in ihren Geschäftsanwendungen. Die in dieser Zeit von SAP neu etablierten Enterprise Services standen während der Entwicklung einer Verbundanwendung oftmals noch nicht zur Verfügung. Zwar wurde die Schnittstelle des Enterprise Service bereits direkt in der Composite verwendet, aber die betriebswirtschaftliche Funktionalität musste oftmals über den Aufruf von RFC-Bausteinen (Remote Function Call – SAP's Implementierung von Remote Procedure Calls (RPC) über in ABAP implementierte Remote Function Modules (RFM)) realisiert werden. Damit zu einem späteren Zeitpunkt von dem RFC-Baustein auf den dann fertigen Enterprise Service gewechselt werden konnte, sorgte der BAL für die nötige Abstraktion. Wie Abbildung 1 zudem entnommen werden kann, sieht die Architektur einen separaten Entwicklungsschritt in den Backend-Systemen zur Bereitstellung der Geschäftslogik in Form von Enterprise

Services vor (Service Enablement). In dieser Arbeit wird hingegen von einem nicht-invasiven Ansatz ausgegangen, der keine Anpassungen in den Altsystemen vorsieht. Zudem ist die in Schuller 2007a beschriebene Architektur stark auf SAP Technologien und SAP Geschäftsanwendungen ausgerichtet und daher nur bedingt für andere Einsatzszenarien geeignet. In Summe lassen sich folgende Unterschiede zwischen der Architektur aus Schuller 2007a und der in dieser Arbeit behandelten Architektur festhalten:

1. Starke Ausrichtung auf Web Services
2. Direkte Aufrufe der Backend-Services aus der Verbundanwendung heraus
3. Ausschließlich synchrone Aufrufe aus der Composite heraus; keine explizite Ausarbeitung asynchroner Szenarien
4. Starke Fokussierung auf SAP-Technologien und -Geschäftsanwendungen
5. Direkte Verwendung der äußerst komplexen Enterprise Service-Schnittstellen in der Verbundanwendung
6. Eingriffe in den Backend-Systemen zur Service-Bereitstellung notwendig

Zu all diesen Punkten wird im Laufe der Arbeit Stellung bezogen und alternative Lösungen unter Berücksichtigung der eingangs erwähnten Ziele vorgeschlagen. Erst in jüngeren Veröffentlichungen (SAP 2009c, Stiehl 2009e, Stiehl 2010a, 2010b, 2010c) wird eine generischere Architektur vorgestellt, die sich in großen Teilen mit den in dieser Arbeit beschriebenen Ansätzen deckt. Schlussendlich berücksichtigt das Papier von Fildebrandt et. al. 2010 die zuvor genannten Ergebnisse, so dass die lose Kopplung auch in offizielle SAP-Papiere Einzug fand (Fildebrandt et. al. 2010, S. 21).

Wie aus der Einleitung bekannt, sollen Composite Applications innerhalb der Unternehmen die Geschäftsprozesse abdecken, die ihnen einen Wettbewerbsvorteil bringen. Von daher unterscheiden sie sich von reinen Integrationsanwendungen, die primär eine technischen Integrationsfokussierung besitzen, also beispielsweise für eine optimierte Kommunikation zwischen dem Unternehmen und seinen Zulieferern und Partnern sorgen. Derartige Lösungen wurden während der Enterprise Application Integration (EAI)-Ära entwickelt und brachten eine neue Generation von Infrastruktur hervor, die für eine reibungslose Abwicklung von einem primär nachrichtenbasierten

Datenaustausch sorgten. Dabei stand weniger die betriebswirtschaftliche als die technische Prozessoptimierung im Mittelpunkt.

Zudem sind Composites von der neuen Generation sogenannter Mashup-Lösungen zu unterscheiden. Mashups sind primär reine Benutzeroberflächen-Programme, die dem Endanwender Bildschirme zur Verfügung stellen, deren Inhalte aus unterschiedlichsten Datenquellen gespeist werden. Oftmals werden auch einfach nur existierende Oberflächen wiederverwendet, die in einem gemeinsamen semantischen Zusammenhang stehen, und zu einem neuen UI verknüpft, wobei jeder wiederverwendete Bildschirm in einem eigenen Teilfenster der Gesamtoberfläche erscheint.

In der Regel ist jedem Bildschirmausschnitt bzw. jedem Teilfenster genau eine Datenquelle zugeordnet. Natürlich kann ein Ausschnitt auch mehrere Quellen anzapfen, doch stellt eine solche Architektur eher die Ausnahme dar. Zusammen ergibt ein derart erstellter Bildschirm eine Benutzeroberfläche, die Daten so zusammenträgt, wie der Endanwender sie für seine jeweilige Rolle im Unternehmen gerade benötigt. Man spricht in diesem Zusammenhang auch von aufgabenorientierten Benutzeroberflächen. In Verbundanwendungen werden ebenfalls aufgabenorientierte Oberflächen erstellt. Der fundamentale Unterschied liegt allerdings darin, wie noch zu sehen sein wird, dass den Bildschirmausschnitten in einer Composite Application keine konkreten Systeme sondern Schnittstellendefinitionen zugeordnet werden. Wie diese Schnittstellen anschließend implementiert werden und welche Systeme zur Erbringung der geforderten Funktionalität involviert werden müssen, ist von vornherein nicht festgelegt und ergibt sich aus der Systemlandschaft, in der die Verbundanwendung eingesetzt werden soll. Eine detaillierte Diskussion dieses Themas ist Gegenstand des dritten Kapitels.

Eine Composite unterscheidet sich auch von klassischen Unternehmensanwendungen wie Standardsoftware von SAP, die sich primär den Geschäftsprozessen in Unternehmen widmet, bei denen die Abläufe entweder innerhalb einer Branche oder sogar branchenübergreifend einheitlich ablaufen. Es handelt sich also um allgemeingültige Prozesse, die sich zudem relativ leicht durch Konfiguration an das jeweilige Zielunternehmen in einem begrenzten Rahmen anpassen lassen. Derartige Lösungen arbeiten in der Regel gegen genau eine Datenbank. Der Entwickler hat zudem vollständige Kontrolle über das gewünschte Transaktionsverhalten. Dieser Ansatz ist für standardisierte Prozesse sicherlich richtig. Doch für die kundenindividuellen Abläufe, die den

Unternehmen Wettbewerbsvorteile bringen sollen, stößt ein solcher Ansatz naturgemäß an seine Grenzen, da sich die Anforderungen grundlegend unterscheiden.

Krafzig et al., 2004 arbeiten gleich zu Beginn ihres Buches *Enterprise SOA – Service Oriented Architecture Best Practices* in Abschnitt 1.2 *(Enterprise Software Is a Different Animal)* sehr treffend die Herausforderungen bei der Erstellung von Software im Unternehmensumfeld heraus, weshalb im Folgenden die wesentlichen Aspekte aus diesem Buch zusammengefasst werden. Demnach stehen Architekten von Unternehmensanwendungen vor einer Vielzahl von Herausforderungen, da eine derartige Software eng mit der internen Unternehmensorganisation, deren Prozessen sowie dem Geschäftsmodell des Unternehmens verknüpft ist (Krafzig et al., 2004, S. 3). Unternehmenssoftware unterliegt sowohl abteilungs- als auch unternehmensübergreifenden Beziehungen (S. 3). Folglich muss sich Unternehmenssoftware mit einer Vielzahl von, oftmals auch widersprüchlichen oder gar unklaren, Anforderungen auseinandersetzen und versuchen, diese bestmöglich zu optimieren (S. 3). Erschwerend kommt hinzu, dass sich Anforderungen aufgrund von geänderten Märkten, Organisationsstrukturen oder Geschäftszielen ebenfalls verändern, so dass sie zu einem beweglichen Ziel werden (S. 3/4). Es ist genau die Berücksichtigung aller Aspekte des Unternehmens und des Geschäftes, die die Unternehmenssoftware so hochgradig komplex werden lässt (S. 4).

Betrachtet man zudem Geschäftsprozesse, die sich über mehrere Abteilungen sowie Systeme erstrecken und dabei gleichzeitig die Einbeziehung von Partnern und Lieferanten berücksichtigen, wird das ganze Ausmaß der Ansprüche an solche Softwaresysteme offensichtlich: nicht nur die heterogene Systemlandschaft (Anwendungssysteme und Integrationslösungen) gilt es zu integrieren, auch die dafür verantwortlichen Teams und den damit verbundenen unternehmenspolitischen Strömungen sind höchst unterschiedlich. Nicht zu vergessen sind zudem die nichtfunktionalen Anforderungen wie Antwortzeitverhalten, Skalierbarkeit, Zuverlässigkeit, Verfügbarkeit und Sicherheit.

Eine durchdachte Softwarearchitektur hilft aber auch noch bei einem weiteren Problem: der Neigung von Unternehmenssoftware mit der Zeit komplexer und änderungsunfreundlicher zu werden (siehe auch Josuttis 2008, S. 2/3 und Krafzig et. al. 2004, S. 4/5). Der Software-Architekt hat bei der Vielzahl der IT-Projekte darauf zu

achten, dass zunehmender Komplexität Einhalt geboten wird, damit die Agilität und die Effizienz bei der Umsetzung neuer Anforderungen nicht leidet. Service-orientierte Architekturen, auf denen auch Verbundanwendungen basieren, verbunden mit einer konsequenten Komponentisierung helfen dabei, die Handlungsfähigkeit der IT sicherzustellen.

Aufgrund dessen lassen sich nun Eigenschaften auflisten, die für Verbundanwendungen typisch sind. Im Rahmen dieser Arbeit wird davon ausgegangen, dass sämtliche Eigenschaften erfüllt sein müssen, um als Verbundanwendung zu gelten. Die nun folgende Aufzählung orientiert sich dabei an SAP (2005), Krafzig et al. (2004) sowie Stiehl (2010a, 2010b).

Folgende Definition (in Anlehnung an SAP 2005, S. 10) fasst wesentliche Aspekte bereits zusammen und dient als Einstieg in die Diskussion:

*Verbundanwendungen sind benutzerzentrische betriebswirtschaftliche webbasierte Applikationen, die hochgradig kollaborative und dynamische Geschäftsprozesse unterstützen, wobei die Prozesse Funktions-, System- und Organisationsgrenzen überschreiten, indem sie die von Plattformen und Anwendungen als Dienste bereitgestellten Daten und Funktionalitäten wiederverwenden.*

Ausgehend von dieser Definition lassen sich nun folgende Eigenschaften ableiten, die in dem folgenden Kapitel 3 weiter verfeinert werden:

- Verbundanwendungen sind zunächst einmal eigenständige Applikationen, die neue betriebswirtschaftliche Funktionalitäten abdecken (SAP 2005, S. 10). Sie orientieren sich dabei an hochinnovativen Geschäftsprozessen, die dem Unternehmen einen Vorteil gegenüber seiner Konkurrenz bringen. Es sind diese differenzierenden Prozesse, die Composites von Standardanwendungen à la SAP unterscheiden: es ist nicht das Ziel, existierende Standardfunktionalitäten neu zu implementieren. Stattdessen stehen gewinnbringende Prozesse im Vordergrund, die die Standardprozesse um neue Funktionalitäten erweitern. Treiber und damit Ausgangspunkt der Composite-Defini-

tion sind demnach die Fachabteilungen, die nach Prozessoptimierungen streben. Verbundanwendungen folgen dabei einem leichtgewichtigen webbasierten Ansatz und unterscheiden sich damit von Rich-Client-Applikationen (RCA), bei denen aufwändige Installationen auf Benutzerseite vorausgesetzt werden.

Typische Szenarien für Verbundanwendungen sind (Grobe 2009):

- Unternehmensspezifische Prozesse bzw. Prozesslücken, die von keinen existierenden (Standard-) Applikationen abgedeckt werden
- Ende-zu-Ende Szenarien, die sich über System-, Anwendungs- und Unternehmensgrenzen hinweg erstrecken
- Szenarien mit hohem Kommunikations-, Koordinations- und Kollaborationsaufwand, die bisher mittels Email, Office-Dokumenten wie Word- bzw. Excel-Dateien oder Papier abgewickelt wurden
- Lösungen, die einen erhöhten Bedarf an wieder verwendbaren und einfach zu pflegenden Geschäftsregeln ausweisen
- Szenarien, bei denen die Notwendigkeit besteht, separierte Prozesse, Prozessschritte, Transaktionen, Benutzeroberflächen sowie UI-Technologien zu vereinfachen bzw. neu zu kombinieren
- Prozesse mit sowohl manuellen als auch automatischen Aktivitäten
- Szenarien mit leichtgewichtigen B2B-Integrationen, also ohne aufwändiges Routing und Datenmapping bzw. ohne aufwändige Datentransformationen
- Selbstbedienungsszenarien wie beispielsweise Urlaubs- und Reiseantrag (innerhalb des Unternehmens) oder Bürgerdienste wie Passbeantragung oder Schadensmeldung bei öffentlichen Einrichtungen (außerhalb des Unternehmens)
- Prozesse, bei denen in der Vergangenheit häufig Änderungen festgestellt wurden bzw. bei denen in Zukunft häufige Änderungen zu erwarten sind
- Szenarien mit Echtzeitanforderungen

- Vom Charakter her sind Verbundanwendungen primär benutzerzentrisch und kollaborativ (SAP 2005, S. 12), d.h. die Beteiligung von Anwendern in den Prozessen ist erwünscht. Für Unternehmen ist es nach wie vor das oberste Ziel, eine weitestgehende hochgradige Automatisierung ihrer Prozesse ganz ohne menschliche Beteiligung zu erreichen. Das ist für Composite Applications nicht anders. Allerdings finden sich gerade in den unternehmensspezifischen Prozessen sehr viele Beispiele, bei denen Experten der Fachabteilungen in den Prozessen mitwirken. Dies erfolgt derzeit leider eher unkoordiniert über den Austausch von Emails oder Office-Dokumenten. Derartige Prozesse sind daher ideale Kandidaten zur Optimierung.

- Da Composite Applications die differenzierenden Prozesse adressieren, ist aufgrund des Nachahmungseffekts bei der Konkurrenz mit einer häufigen Änderungsrate (SAP 2005, S. 13) für diese Prozesse zu rechnen: um den Wettbewerbsvorteil langfristig zu sichern, sind Unternehmen ständig gezwungen, ihre Prozesse den ändernden Gegebenheiten anzupassen. Eine hohe Dynamik muss bei der Konzeption von Verbundanwendungen demnach Rechnung getragen werden.

- Verbundanwendungen überschreiten System-, Anwendungs- und Unternehmensgrenzen (SAP 2005, S. 10), ein im Vergleich zu anderen Lösungen fundamentaler Unterschied: es liegt in der Natur von Composites, Grenzen zu überschreiten, da gerade die Abdeckung kompletter Wertschöpfungsketten wie Bestell- und Zahlungsprozesse (Order-to-Cash) oder Beschaffungsprozesse (Procure-to-Pay) in deren Mittelpunkt stehen. Zwangsläufig sind unterschiedlichste Systeme innerhalb und außerhalb des Unternehmens davon betroffen, auf die Composite Applications vorbereitet sein müssen.

- Verbundanwendungen bauen auf der Wiederverwendbarkeit existierender Funktionalitäten auf (SAP 2005, S. 12). Ohne diese Voraussetzung ist ihre Existenz undenkbar. Es ist eine der Grundideen von Composite Applications, dass sie auf bereits vorhandener fachlicher Logik aufbauen, um sie zu neuen Prozessen zusammenzufügen. Genau an diesem Punkt greift die

SOA-Idee und die Komponentisierung von Fachlichkeit. Gemeinsam stellen sie die Voraussetzungen für den Bau von Composites dar. Derartige Komponenten, ausgestattet mit einem abgeschlossenen betriebswirtschaftlichen Funktionsumfang, werden als Web Services bereitgestellt, auf die Verbundanwendungen bei der Realisierung innovativer Lösungen zurückgreifen können.

- Wie die schematische Darstellung einer Verbundanwendung in Abbildung 2 verdeutlicht, folgt eine Composite Application einem Schichtenmodell, das sich aus einer Prozess-, einer Benutzeroberflächen- sowie einer Service- bzw. Geschäftsobjektschicht zusammensetzt (SAP 2005, S. 13/14). Die Prozessschicht behandelt die bereits erläuterten innovativen Geschäftsprozesse. Zu deren Implementierung greifen die Geschäftsprozesse auf entsprechende Benutzeroberflächen bzw. Services und Geschäftsobjekte zurück.

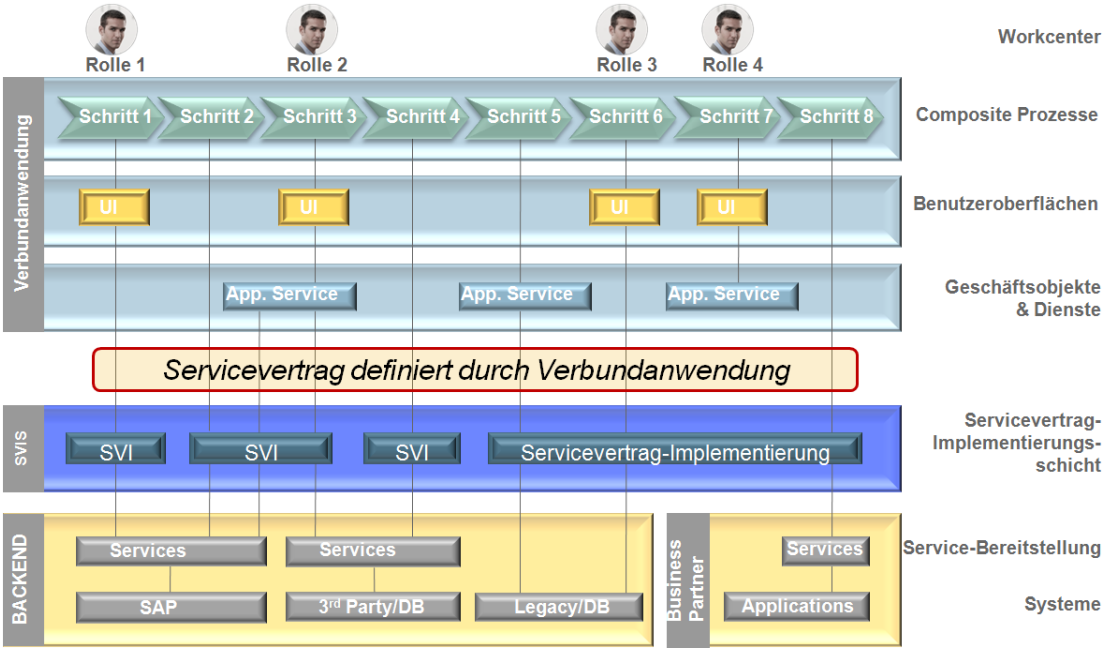



*Abbildung 2: Schematische Darstellung einer Verbundanwendung (Grimm, Speck, Stiehl 2010)*

- Die Endanwendersicht wird durch aufgabenorientierte Benutzeroberflächen repräsentiert (SAP 2005, S. 16). Sie sind speziell auf die Bedürfnisse des Anwenders in seiner jeweiligen Prozessrolle zugeschnitten. Zur Realisie-

rung ihrer Aufgaben bedienen sich die Oberflächen, wie auch schon die Prozessschicht, der Dienste der Geschäftsobjekt- und Serviceschicht.

- Die Geschäftsobjekt- und Serviceschicht stellt sämtliche Services zur Verfügung, die von der Prozess- und Oberflächenschicht zur Erfüllung der angestrebten Fachlichkeit benötigt werden (SAP 2005, S. 15). Dies umfasst die Bereitstellung komplett neuer, composite-spezifischer Geschäftslogik, Datenpersistenz für neue Geschäftsobjekte sowie die Wiederverwendung existierender externer Dienste.

- Verbundanwendungen sind gegenüber den aufgerufenen Backend-Systemen lose gekoppelt (SAP 2005, S. 10), sind demnach also vollkommen unabhängig von ihnen. Die Composite weiß letztendlich nicht, gegen welche Systeme sie konkret läuft bzw. laufen wird. Es ist gerade diese Denkweise, die bei der Entwicklung von Composite Applications die größten Probleme bereitet und soll daher insbesondere in Kapitel 3 weiter detailliert werden.

- Aufgrund ihrer Eigenständigkeit besitzen Verbundanwendungen einen eigenen Lebenszyklus (SAP 2005, S. 10), der vollständig von den Lebenszyklen der involvierten Systeme getrennt ist. Es gibt also keinerlei Abhängigkeiten zwischen den Versionen einer Composite und den Versionen der aufgerufenen Backend-Systeme. Folglich ist eine Composite gegen Releasewechsel der integrierten Anwendungen immun. Erreicht wird dies durch eine konsequente Trennung der Verbundanwendung von den zu integrierenden Systemen mit Hilfe einer separaten Schicht, dem sogenannten Service Contract Implementation Layer (SCIL – Stiehl 2010a) bzw. der Servicevertrag-Implementierungsschicht. Wie das Zusammenspiel zwischen Verbundanwendung und SCIL im Detail funktioniert ist abermals Gegenstand des dritten Kapitels.

- Verbundanwendungen arbeiten ausschließlich auf den Daten, die sie konkret zur Umsetzung ihrer fachlichen Anforderungen benötigen (Stiehl 2010a). Der Architekt wird bei der Planung des Datenmodells, auf dem die

Composite operiert, von dem/den zu implementierenden Geschäftsprozess/en getrieben. Gemäß diesen Anforderungen identifiziert er die benötigten Geschäftsobjekte mit ihren Ausprägungen, also benötigten Attributen. Ziel ist es, die Anzahl der Attribute eines Geschäftsobjekts auf ein Minimum zu reduzieren.

- Verbundanwendungen arbeiten auf einem kanonischen Datentypsystem und erreichen somit auf Datentypebene eine lose Kopplung zu ihrer Umwelt (Stiehl 2010a). Sie verzichten bewusst auf die Wiederverwendung existierender Datentypen, die beispielsweise in den verwendeten Backendsystemen vorhanden sind, da diese ihre Unabhängigkeit verletzen würden, was es unter allen Umständen zu vermeiden gilt. In Kapitel 3 wird eine weiterführende Diskussion dieses Punktes folgen.

- Verbundanwendungen sind nicht invasiv (Stiehl 2010a). Der Einsatz einer Composite sollte demnach keinerlei Anpassungen in den angeschlossenen Systemen zur Folge haben, um von der Funktionalität einer Composite Application profitieren zu können. Im Umkehrschluss bedeutet dies, dass Services in den zu integrierenden Systemen so zu verwenden sind, wie sie angeboten werden.

Aufgrund der aufgeführten Eigenschaften lassen sich nun die folgenden drei Kernziele von Composites zusammenfassen (Banerjee 2006):

1. Alignment: Ausrichtung nach und Abstimmung mit den Hauptakteuren.
   Die von der Geschäftsleitung vorgegebene Geschäftsstrategie muss durch die zeitnahe Implementierung durch Composite Applications nahtlos umgesetzt werden können. Composites müssen dazu einerseits die verfügbaren IT-Ressourcen mit den Bedürfnissen der Fachabteilungen in Einklang bringen. Andererseits erfüllen sie die Aufgabe, unterschiedlichste Gruppen, Abteilungen und Organisationen innerhalb und außerhalb eines Unternehmens zusammenzubringen. Dies wird durch die Implementierung ganzheitlicher Ende-zu-Ende-Prozesse und durch Kollaborationsfunktionen innerhalb der

Software erreicht.

2. Adaptibility: Sollten sich die Marktgegebenheiten ändern, so muss eine Verbundanwendung auf die neuen Anforderungen schnellstmöglich angepasst werden können. Das bedingte eine Auslagerung variabler Bestandteile in separate, leicht änderbare Komponenten, wie dies beispielsweise bei Regelwerken der Fall ist. Umgekehrt muss durch geeignete Rekonfigurationsvorkehrungen in der Verbundanwendung selbst eine erhöhte Anpassungsfähigkeit gewährleistet werden. Die Anpassbarkeit ist dabei also eine Fähigkeit der Anwendung selbst, sich ohne zusätzliche Programmierung durch die angesprochenen Maßnahmen auf geänderte Bedingungen einstellen zu lassen.

3. Agility: Eine erhöhte Agilität einer Verbundanwendung gewährleistet eine reduzierte Zeit, bis die Entwicklungskosten durch Einsparungen aufgrund einer effizienteren Prozessabwicklung amortisiert sind. Erreicht wird dies beispielsweise durch die Verwendung von Industriestandards, Komponenten mit wohldefinierter, also in sich abgeschlossener betriebswirtschaftlicher Logik, etablierten Best-Practices bzw. Pattern während der Entwicklung, die eine effizientere Umsetzung, aber auch einfacheres Deployment und Konfigurierung von Verbundanwendungen zur Folge haben. Agilität bezieht sich in diesem Zusammenhang auf den Entwicklungsprozess und ist dadurch zur Anpassbarkeit (Adaptability) zu unterscheiden. Auch hier wird letztendlich das Ziel verfolgt, auf geänderte Marktgegebenheiten reagieren und die Vorgaben des Managements schnellstmöglich umsetzen zu können.

Bevor im Folgenden ausgewählte Beispiele zeigen sollen, welchen Funktionsumfang die Prozesse von Verbundanwendungen typischerweise abdecken, wird im nächsten Unterkapitel die fundamentale Rolle der standardisierten Notation zur grafischen Modellierung von Prozessen BPMN im Zusammenhang mit Verbundanwendungen detailliert diskutiert.

## 2.1 Die Rolle der BPMN (Business Process Model and Notation) für Verbundanwendungen - Grundlagen

Bereits zu diesem Zeitpunkt muss auf die wichtige Rolle der BPMN für Verbundanwendungen hingewiesen werden. Dies ist nicht nur darauf zurückzuführen, dass in dieser Arbeit sämtliche Prozessmodelle mit der BPMN umgesetzt werden, sondern weil insbesondere mit der neusten Version 2.0 der Spezifikation auch deren Ausführbarkeit zu einem wichtigen Bestandteil geworden ist. Letztendlich wird BPMN somit zu einer Implementierungssprache sowohl von betriebswirtschaftlichen aber eben auch von technischen Prozessen. Während sich in der Literatur (z.B. Stein 2009 oder Lautenbacher 2009) noch immer primär an den ereignisgesteuerten Prozessketten (EPK) für die fachlichen Prozesse und der Business Process Execution Language (BPEL) für die technischen Prozesse orientiert wird, mit entsprechend schwierigen Transformationen zwischen diesen beiden Welten als Folge, setzt diese Arbeit vollständig auf BPMN und vermeidet auf diese Weise jegliche Verluste, die durch Transformationsschritte zwischen unterschiedlichen Notationen auftreten können.

An dieser Stelle wird daher jetzt eine kurze Einführung in diesen Standard gegeben, sofern sie für das weitere Verständnis der in dieser Arbeit verwendeten BPMN-Modelle notwendig ist. Für detailliertere Einführungen in den BPMN-Standard sei auf Allweyer 2009a bzw. Freund, Rücker & Henninger (2010) verwiesen. Zu einem späteren Zeitpunkt wird zudem vertieft auf die besondere Bedeutung der BPMN für Verbundanwendungen eingegangen.

BPMN (OMG 2010a) wurde seitens der Business Process Management Initiative (BPMI) mit dem Ziel einer grafischen Notation zur Darstellung von Prozessbeschreibungen entworfen. Diese Notation sollte darüber hinaus sowohl von der fachlichen als auch von der technischen Seite verwendet und verstanden werden. Aufschluss darüber gibt das folgende Zitat aus der BPMN 2.0-Spezifikation (OMG 2010a):

The primary goal of BPMN is to provide a notation that is readily understandable by all business users, from the business analysts that create the initial drafts of the processes, to the technical developers responsible for implementing the technology that will perform those processes, and finally, to the business people who will manage

and monitor those processes. Thus, BPMN creates a standardized bridge for the gap between the business process design and process implementation.

Im Kern werden mit BPMN unterschiedlichste Diagramme entworfen. In der Version 2.0 können folgende Diagrammtypen unterschieden werden:

- Sequenzdiagramm
- Kollaborationsdiagramm
- Choreographiediagramm
- Konversationsdiagramm

Der für die Beschreibung von Prozessabläufen wichtigste Diagrammtyp ist das Sequenzdiagramm, in dem konkret modelliert wird, welche Aktivitäten in welcher Reihenfolge wann und unter welchen Bedingungen ausgeführt werden. Dabei wird die Ausnahmebehandlung ausdrücklich mit berücksichtigt. Aufgrund seiner Bedeutung, werden die wesentlichen Kernelemente dieses Diagrammtyps im weiteren Verlauf dieses Abschnitts noch detaillierter erläutert.

Eine Besonderheit der BPMN gegenüber anderen Prozessnotationen ist die Möglichkeit zur Modellierung des Zusammenspiels zwischen zwei oder mehreren Prozessen basierend auf Nachrichtenaustausch. Zur Veranschaulichung werden dafür Kollaborationsdiagramme verwendet. Auch sie werden in dieser Arbeit häufiger zu finden sein, da Verbundanwendungen auf die Zusammenarbeit mit ihrem Umfeld angewiesen sind.

Die beiden letztgenannten Diagrammtypen, Choreographiediagramm und Konversationsdiagramm, spielen in dieser Arbeit keine Rolle. Choreographiediagramme stellen die zeitliche und logische Folge der zwischen Prozessen ausgetauschten Nachrichtenflüsse im Vergleich zu Kollaborationsdiagrammen detaillierter dar. Diese Details sind für die Architekturdiskussion allerdings unerheblich und können daher vernachlässigt werden. Bei den Konversationsdiagrammen handelt es sich um Übersichtsdiagramme, die in komplexeren Prozesslandschaften, insbesondere auch unter Berücksichtigung externer Partner, eine kompakte Visualisierung des Gesamtszenarios gewährleisten. Gerade für die unternehmensübergreifende Zusammenarbeit sind diese Diagramme nützlich. Allerdings werden die Szenarien in dieser Arbeit nicht die Kom-

plexität erreichen, als dass sie den Einsatz von Konversationsdiagrammen rechtfertigen würden.

### 2.1.1 BPMN-Kernelemente

Im Mittelpunkt der BPMN-Modellierung stehen die Prozessdiagramme, die auch als Business Process Diagram (BPD) bezeichnet werden und grafisch festlegen, was in welcher Reihenfolge wann unter bestimmten Bedingungen auszuführen ist, wobei es stets zu besonderen Ausnahmesituationen kommen kann, deren Behandlung bei der Modellierung ebenfalls zu berücksichtigen ist. Zur Umsetzung der im letzten Satz genannten Eigenschaften von BPDs sieht die BPMN fünf Kategorien von Kernelementen vor:

1. Fluss-Objekte: sie stellen die grafischen Hauptelemente der BPMN dar und definieren das Verhalten eines Prozesses. Dabei werden drei Flussobjekte unterschieden:
   - Ereignisse
   - Aktivitäten
   - Gateways

2. Daten: dabei handelt es sich um Informationen, die entweder innerhalb eines Prozesses verarbeitet werden oder zwischen verschiedenen Prozessen ausgetauscht werden. Hierunter fallen die folgenden vier BPMN-Elemente:
   - Datenobjekt
   - Eingabedaten
   - Ausgabedaten
   - Datenspeicher

3. Verbindende Objekte: sie erlauben die Verbindung der Fluss-Objekte untereinander aber auch die Verbindung zu ergänzenden Informationen. Die folgenden vier verbindenden Objekte sind hier zu unterscheiden:
   - Sequenzfluss
   - Nachrichtenfluss
   - Assoziation

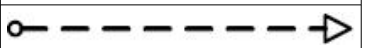
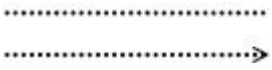
- Daten-Assoziation

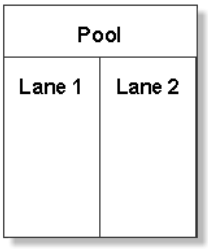
4. Schwimmbahnen: durch sie werden die zuvor genannten primären Modellierungselemente gruppiert. Dazu gibt es in der BPMN zwei Arten der Gruppierung:
   - Pools
   - Bahnen

5. Artefakte: über sie wird ein Prozess mit zusätzliche Informationen versehen. Derzeit umfassen die Artefakte zwei Elemente:
   - Gruppe
   - Textanmerkungen

Die folgende Tabelle (Tabelle 1 nach OMG 2010a, S. 59ff) fasst die genannten Kernelemente mit ihren wesentlichen Eigenschaften zusammen:

| Element | Beschreibung | Notation |
|---|---|---|
| Ereignis | Ein Ereignis drückt aus, dass während des Prozessablaufs *irgendetwas* passiert ist. Ereignisse beeinflussen den weiteren Prozessablauf und haben entweder eine Ursache (auch *Trigger* genannt), der sie ausgelöst hat, oder sie reflektieren ein Ergebnis, dass nach außen weitergereicht wird. Dementsprechend unterscheidet man empfangende oder sendende Ereignisse. Ereignisse werden als Kreise dargestellt. Zudem gibt es drei Arten von Ereignissen: Start-, Zwischen- und Ereignisse. Das Startereignis wird als Kreis mit einfacher Linie dargestellt und gibt an, wo ein Prozess beginnt. Beim Zwischenereignis wird die einfache Linie um eine Zweite ergänzt und tritt im Sequenzdiagramm stets zwischen Start- und Endereignis auf. Das Endereignis schließlich besitzt einen dicken Rand (siehe auch nebenstehende Notation) und zeigt an, wo ein Prozess endet. Die Grundelemente können je nach Ursache oder Ergebnis noch mit passenden Icons gefüllt werden. |  |

| | | |
|---|---|---|
| Aktivität | Eine Aktivität ist eine generische Bezeichnung für Aktionen, die während des Prozessablaufs zu erledigen sind. Dabei kann eine Aktivität entweder atomar oder nicht-atomar (d.h. aus mehreren atomaren Aktivitäten zusammengesetzt) sein. Dementsprechend gibt es zwei Aktivitätstypen: die Task (Aufgabe) und den Unterprozess, die beide durch ein Rechteck mit abgerundeten Ecken symbolisiert werden. |  |
| Gateway | Gateways werden zur Kontrolle der Verzweigungen und Zusammenführungen von Sequenzflüssen innerhalb eines Prozessdiagramms verwendet. Es legt folglich fest, unter welchen Bedingungen gewissen Pfaden gefolgt wird bzw. wie Pfade zusammengeführt werden. Sie werden durch Rauten dargestellt, wobei spezielle Markierungen innerhalb der Rauten unterschiedliche Gateway-Typen und damit unterschiedliches Verhalten an den Gateways ausdrücken. |  |
| Sequenzfluss | Ein Sequenzfluss wird eingesetzt, um die Reihenfolge festzulegen, in der Aktivitäten innerhalb eines Prozesses ausgeführt werden. Er wird durch einen gerichteten Pfeil dargestellt. |  |
| Nachrichtenfluss | Nachrichtenflüsse veranschaulichen den Nachrichtenaustausch zwischen zwei oder mehreren Prozessen. Auf diese Weise wird die Zusammenarbeit der Prozesse dargestellt. Die Richtung des Nachrichtenaustauschs wird ebenfalls durch einen gerichteten Pfeil dargestellt, der diesmal jedoch gestrichelt ist. |  |
| Assoziation | Die Assoziation wird benutzt, um ergänzende Informationen und Artefakte mit den grafischen BPMN-Elementen zu verbinden. Dabei handelt es sich um gepunktete Linien, die entweder gerichtet oder ungerichtet sein können, je nachdem, was mit dem BPMN-Element verbunden wird. So können für Datenobjekte durch gerichtete Assoziationen die Datenflüsse visualisiert werden. Textuelle Anmerkungen hingegen sind ungerichtet assoziiert. |  |
| Pool | Ein Pool ist zunächst einmal ein Container für einen vollständigen Prozess, der folglich den gesamten Sequenzfluss umfasst. Pools spielen aber auch in Kollaborationsdiagrammen eine wichtige Rolle, um das Zusammenspiel mehrerer Pro- |  |

| | | |
|---|---|---|
| | zesse darzustellen. Dabei repräsentiert ein Pool grafisch einen Teilnehmer in einer solchen Kollaboration. Da es bei Kollaborationsdiagrammen hauptsächlich auf den Nachrichtenaustausch zwischen den Teilnehmern ankommt, können die Internas eines Pools, also dessen Sequenzfluss, ausgeblendet werden. Er wird dann zu einer *Black Box*, im Gegensatz zur *White Box*, die sämtliche Details eines Prozesses darstellt. | |
| (Schwimm-)Bahn | Eine (Schwimm-)Bahn sorgt für eine weitere Unterteilung innerhalb eines Pools und wird zur Organisation und Kategorisierung der in ihr enthaltenen Aktivitäten benutzt. Beispielsweise werden Bahnen zur Zuweisung von Rollen bzw. Unternehmens-Organisationseinheiten verwendet: sämtliche Aktivitäten, die einer solchen Bahn zugewiesen werden, sind von dieser Rolle bzw. Organisationseinheit durchzuführen. Analog kann eine Bahn auch ein IT-System oder eine Anwendung abbilden, die die Ausführung gewisser Aktivitäten übernimmt. Pools und Bahnen können sowohl vertikal als auch horizontal dargestellt werden. |  <br>  |
| Datenobjekt | Datenobjekte umfassen Informationen über die Daten, die Aktivitäten zur Bewältigung ihrer Aufgabe benötigen. Sie umfassen sowohl Eingabe- als auch Ausgabedaten. Datenobjekte können einfache Felder, komplexe Strukturen oder Listen von Objekten sein. |  |
| Nachricht | Eine Nachricht wird in Kollaborationsszenarien verwendet und stellt den Inhalt der zwischen den beteiligten Prozessen ausgetauschten Informationen dar. |  |
| Gruppe (ein Rechteck um eine Gruppe von Objekten) | Eine Gruppe fasst BPMN-Elemente aus Dokumentations- oder Analysegründen zusammen. Die Gruppierung hat keinerlei Auswirkung auf den Sequenzfluss innerhalb dieser Gruppe. |  |

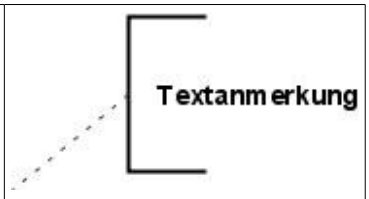
| | | |
|---|---|---|
| Textanmerkung (über eine Assoziation verbunden) | Textanmerkungen erlauben dem Modellierer, seinem Modell zur besseren Lesbarkeit ergänzende textuelle Informationen hinzuzufügen. Diese Texte können über Assoziationen mit BPMN-Elementen verbunden sein, um zu verdeutlichen, auf welches BPMN-Element sich der Text bezieht. |  |

***Tabelle 1: Kernelemente der BPMN 2.0 (OMG 2010a)***

#### 2.1.1.1 Semantik von Prozessmodellen

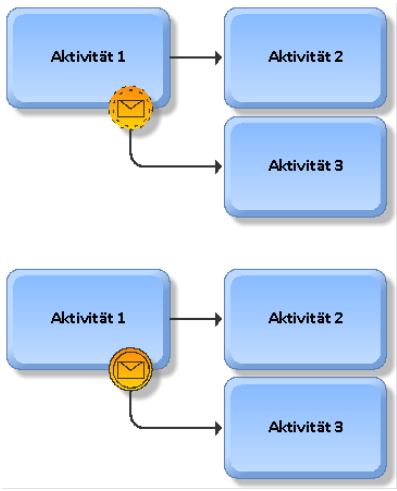
Bevor weitere Details von Ereignissen, Gateways und Aktivitäten erläutert werden können, muss kurz auf die Semantik von Prozessmodellen eingegangen werden. Dazu wird in der Spezifikation das Konzept der Token oder Marken verwendet, die durch ein Prozessmodell wandern (OMG 2010a, S. 57). So wird beim Start eines Prozesses eine entsprechende Marke auf dem dazugehörigen Startereignis erzeugt und wandert von dort gemäß des modellierten Prozessflusses zu den verschiedenen Aktivitäten, Gateways und Ereignissen. Erreicht das Token beispielsweise eine Aktivität, verharrt es solange auf der Aktivität, bis die dazugehörige Aufgabe bearbeitet wurde. Bei Gateways verzweigt das Token und kann sich, je nach Gateway-Typ, sogar vermehren, so dass letztendlich mehrere Pfade parallel ausgeführt werden. Analog können Token an Gateways, die verschiedene Prozesspfade zusammenführen, wieder gebündelt und als ein Token weitergegeben werden, so dass sich deren Anzahl wieder reduziert. Das konkrete Verhalten hängt allerdings, wie schon erwähnt, vom konkreten Gateway-Typ ab und wird in Abschnitt 2.1.1.3 noch detaillierter betrachtet. Letztendlich erreichen die Marken die Endereignisse, wo sie schließlich vernichtet werden. Befindet sich kein Token mehr im Umlauf, ist der Prozess beendet.

Die Einführung von Marken dient allerdings lediglich als theoretisches Konzept zur Diskussion des Prozessverhaltens. Firmen, die eine BPMN-Implementierung anbieten, müssen dieses Konzept nicht entsprechend in Programmcode abbilden (OMG 2010a, S. 57). Lediglich die durch die Marken verdeutlichte Semantik muss passend umgesetzt werden.

#### 2.1.1.2 Ereignisse

Bei der Beschreibung der Ereignisse wurde bereits auf deren Vielfalt hingewiesen. Es ist eine der Stärken von BPMN, im Modell auf unterschiedlichste Situationen mit Hilfe von Ereignissen reagieren zu können. Damit Prozessmodelle mit Ereignissen

besser zu verstehen sind, verdeutlicht die nachfolgende Tabelle, in welcher Form Ereignisse in Modellen eingesetzt werden können und wie die Funktionsweise richtig zu interpretieren ist. Am Beispiel des Nachrichtenereignisses werden die verschiedenen Fälle erläutert. Ein Nachrichtenereignis stellt dabei eine spezielle Ereignisform dar und ist an einem Briefumschlag (  ) innerhalb des Kreises, der generell ein Ereignis symbolisiert, zu erkennen. Dabei repräsentiert der Briefumschlag die Ursache des Ereignisses, nämlich eine Nachricht. So wie der Umschlag eine Nachricht als Ereignisauslöser charakterisiert, stellt eine Uhr (  ) ein Timer-Ereignis dar. Insgesamt unterscheidet BPMN 2.0 13 verschiedene Ereignistypen.

| Element | Beschreibung | Notation |
|---|---|---|
| Empfangende Nachrichtenereignisse | Empfangende Ereignisse sind an den ungefüllten bzw. nicht mit schwarzer Farbe gefüllten Symbolen zu erkennen. Für das Nachrichtenereignis heißt das konkret, dass der Inhalt des Briefumschlags nicht schwarz gefüllt ist (siehe Notation). Empfangende Ereignisse können als Start- oder Zwischenereignis eingesetzt werden. Der Einsatz als Startereignis bedeutet, dass auf das Eintreffen einer Nachricht zum Start des Prozesses gewartet wird. Analog wird beim Zwischenereignis der Prozessfluss unterbrochen, bis eine entsprechende Nachricht eingetroffen ist. |  |
| Sendende Nachrichtenereignisse | Sendende Ereignisse sind an den mit schwarzer Farbe gefüllten Symbolen zu erkennen. Das bedeutet konkret für das Nachrichtenereignis, dass der Inhalt des Briefumschlags schwarz gefüllt ist. Sendende Ereignisse können als End- oder Zwischenereignis eingesetzt werden. Der Einsatz als Endereignis bedeutet, dass bei Prozessende noch eine Nachricht verschickt wird. Analog wird beim Zwischenereignis eine entsprechende Nachricht verschickt und der Prozess unmittelbar mit dem nächsten Schritt im Prozessfluss fortgesetzt. |  |
| Angeheftete Nachrichtenereignisse | Neben dem Einsatz der Ereignisse direkt im Prozessfluss, können sie auch an Aktivitäten angeheftet werden. Dadurch wird zum Ausdruck gebracht, dass bei Eintreten des entsprechenden Ereignisses *während* der Bearbeitung der Aktivität, an dem sich das Ereignis befindet, dem Prozessfluss gefolgt wird, der dem angehefteten Ereignis zugeordnet ist. So wird in nebenstehender Abbildung jeweils Aktivität 3 ausgeführt, sobald während der Bearbeitung von Aktivität 1 eine Nachricht eintrifft. Dabei wird Aktivität 1, bei der das angeheftete Ereignis einen gestrichelten Rand besitzt, nicht unterbrochen, während |  |

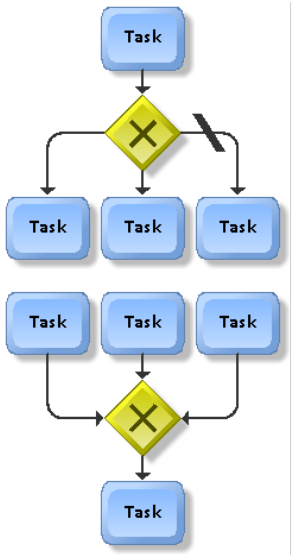
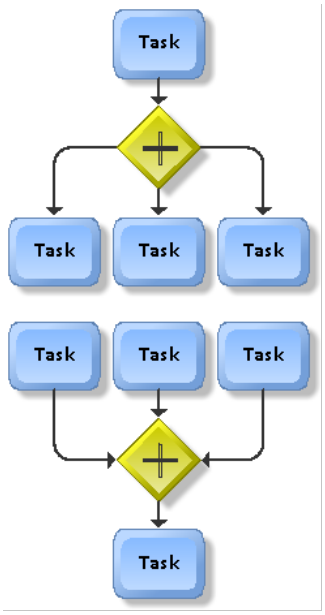
| | | |
|---|---|---|
| | das Nachrichtenereignis mit dem durchgezogenen Rand dafür sorgt, dass bei Eintreffen einer Nachricht die ihm zugeordnete Aktivität unterbrochen wird. | |

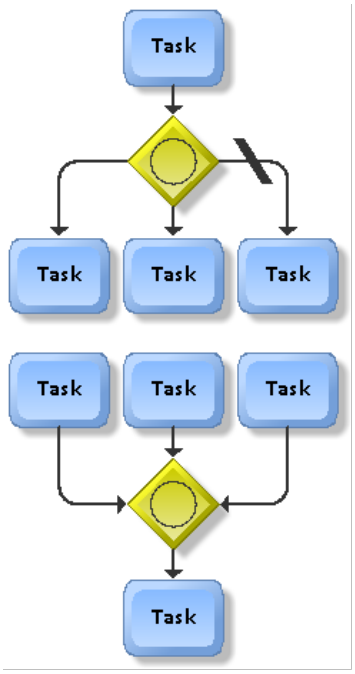
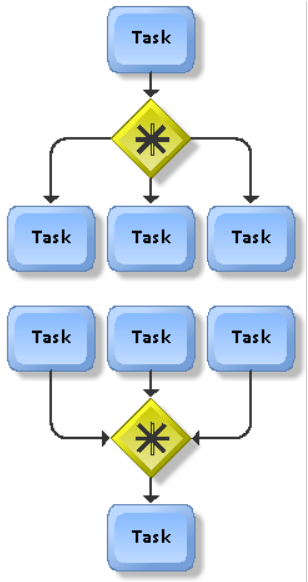
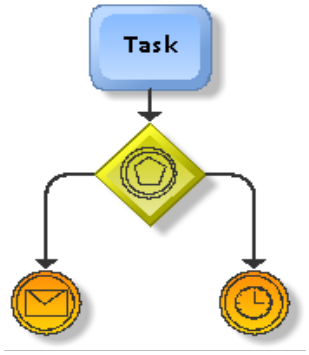
***Tabelle 2: Ereignisse der BPMN 2.0 (OMG 2010a)***

Durch Kombination der in Tabelle 2 dargestellten Einsatzmöglichkeiten von Ereignissen lassen sich nun vielfältige Fehlersituationen modellieren, wie im Laufe dieser Arbeit auch noch sehen sein wird (Kapitel 4.1.3 und 5.4). Wichtig ist an dieser Stelle die Unterscheidung zwischen Start-, Zwischen- und Endereignissen sowie zwischen sendenden, empfangenden, angehefteten nicht-unterbrechenden bzw. angehefteten unterbrechenden Ereignissen. Mit diesem Wissen sind die nun folgenden Prozessmodelle gut zu verstehen. Auf die verschiedenen Ereignistypen wird je nach Bedarf kontextabhängig eingegangen.

#### 2.1.1.3 Gateways

Ereignisse ermöglichen bereits eine recht differenzierte Behandlung außergewöhnlicher Prozesssituationen und sind daher ein wichtiger Bestandteil der Notation. Von ebenso wichtiger Bedeutung sind Gateways, die für eine Aufteilung und Zusammenführung von Prozessflüssen sorgen. BPMN unterscheidet fünf Gateways, die innerhalb von Prozessen und zwei Gateways, die zum Start von Prozessen eingesetzt werden können. Die nun folgende Tabelle 3 erklärt die Unterschiede:

| Element | Beschreibung | Notation |
|---|---|---|
| Exklusives Gateway | Beim exklusiven Gateway, zu erkennen an dem **X** innerhalb der Raute, folgt der Prozessfluss bei einer Verzweigung genau einem ausgehenden Pfad. Jeder Pfad muss daher mit einer Bedingung versehen sein. Die Bedingungen müssen sich zudem gegenseitig ausschließen. Folglich wird auch nur *eine* Marke am Ausgang der Gateways weitergereicht. Trifft keine der angegebenen Bedingungen ein, so wird dem Default-Zweig gefolgt, zu erkennen an einem Querstrich durch den dazugehörigen Sequenzfluss.<br><br>Wird das exklusive Gateway zur Zusammenführung eingesetzt, so wird unmittelbar nach Eintreffen einer Marke aus einem der eingehenden Pfade ein entsprechendes Token am Ausgang weitergereicht. Folglich wird die dem Gateway folgende Aktivität unter Umständen mehrfach ausgeführt. |  |
| Paralleles Gateway | Beim parallelen Gateway, zu erkennen an dem **+** innerhalb der Raute, wird unabhängig von Bedingungen an jedem ausgehenden Pfad eine Marke weitergereicht. Dadurch werden parallele Prozessflüsse gestartet und es findet eine Markenvermehrung statt.<br><br>Wird das parallele Gateway zur Zusammenführung eingesetzt, so wird so lange an dem Gateway gewartet, bis aus allen eingehenden Sequenzflüssen Marken eingetroffen sind. Erst dann wird genau ein Token am Ausgang weitergereicht. Es findet also eine Markenreduzierung statt. |  |

| | | |
|---|---|---|
| Inklusives Gateway | Beim inklusiven bzw. Oder-Gateway, zu erkennen an dem **O** innerhalb der Raute, folgt der Prozessfluss bei einer Verzweigung den ausgehenden Pfaden, deren Bedingung erfüllt ist. Jeder Pfad muss daher mit einer Bedingung versehen sein. Folglich können ein oder mehrere Token am Ausgang des Gateways weitergereicht werden. Auch hier ist die Modellierung eines Default-Pfades möglich, dem gefolgt wird, sofern keine der angegebenen Bedingungen erfüllt ist.<br><br>Wird das inklusive Gateway zur Zusammenführung eingesetzt, so wird so lange an dem Gateway gewartet, bis sämtliche Marken eingetroffen sind, die das Gateway noch erreichen können. Das Gateway ist folglich darüber informiert, wie viele Token aufgrund der vorangegangenen Prozesslogik erzeugt wurden und noch zum Gateway gelangen können. Erst dann wird genau ein Token am Ausgang weitergereicht. Es findet also eine Markenreduzierung statt. |  |
| Komplexes Gateway | Beim komplexen Gateway, zu erkennen an dem Stern (✱) innerhalb der Raute, können sowohl bei der Verzweigung als auch bei der Zusammenführung beliebige Bedingungen hinterlegt werden, wie mit den Marken zu verfahren ist. Dadurch werden letztendlich die Fälle abgedeckt, die mit den anderen Gateways nicht umsetzbar sind. Beispielsweise können auf diese Weise n-aus-m-Zusammenführungen (mit n <= m) realisiert werden. Ein Beispiel für eine solche Regel sähe wie folgt aus: “Wenn 3 von 5 Pfaden beendet wurden, soll der Prozess fortgesetzt werden und sämtliche danach folgenden Token sollen ohne Weitergabe einfach eliminiert werden“. |  |
| Ereignisbasiertes exklusives Gateway | Das ereignisbasierte exklusive Gateway verzweigt den Prozessfluss abhängig von eintreffenden Ereignissen. Dabei wird das Gateway mit mehreren Ereignissen verbunden. Es wird nun dem Sequenzfluss gefolgt, dessen Ereignis als erstes eintrifft. Im Beispiel rechts wird auf eine Nachricht oder dem Ablauf einer Zeitüberwachung gewartet. Je nachdem, ob zuerst die Nachricht eintrifft oder der Timer abläuft, wird dem entsprechenden Pfad gefolgt. |  |

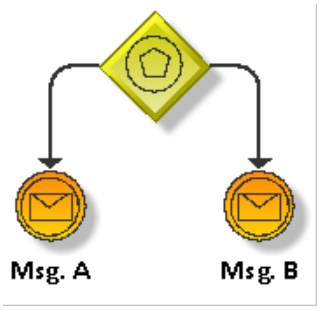
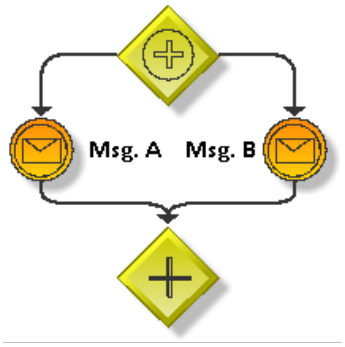
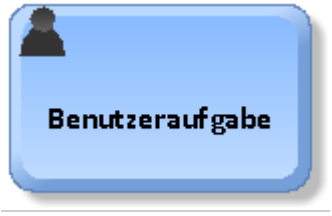
| | | |
|---|---|---|
| Instanziierendes exklusives ereignisbasiertes Gateway | Das ereignisbasierte Gateway kann in einer speziellen Abwandlung auch zum Instanziieren von Prozessen eingesetzt werden. Der Start des Prozesses hängt hier wiederum von *einem* der dem Gateway folgenden Ereignisse ab. Sobald eines von Ihnen eintrifft, wird der Prozess gestartet und dem Sequenzfluss gefolgt, der zu dem eingetroffenen Ereignis gehört. Auf das Eintreffen des zweiten Ereignisses wird anschließend nicht mehr gewartet. |  |
| Instanziierendes paralleles ereignisbasiertes Gateway | Das instanziierende parallele ereignisbasierte Gateway ist eine weitere Variante des instanziierenden ereignisbasierten Gateways. Auch hier wird der Prozess gestartet, sobald das erste der dem Gateway folgenden Ereignisse gefeuert wird. Allerdings bleiben nach Eintreffen eines Ereignisses die jeweils noch nicht eingetretenen Ereignisse nach wie vor aktiv, so dass weiterhin auf deren Ankunft gewartet wird. Werden die Zweige, wie in der nebenstehenden Abbildung gezeigt, zudem über das parallele Gateway zusammengeführt, müssen folglich alle Ereignisse gefeuert worden sein, ehe der Prozess fortgesetzt wird. Im Beispiel rechts müssen demnach als Voraussetzung für die Prozessfortsetzung sowohl Nachricht A als auch Nachricht B eingetroffen sein. |  |

***Tabelle 3: Gateways der BPMN 2.0 (OMG 2010a)***

#### 2.1.1.4 Aktivitäten

Wie bereits aus Abschnitt 2.1.1 bekannt ist, sind bei den Aktivitäten sowohl Aufgaben (atomare Aktivitäten) als auch Unterprozesse (nicht-atomare Aktivitäten) unterscheidbar. Aufgaben können nun wiederum weiter verfeinert werden. Tabelle 4 fasst die wesentlichen Aufgabentypen mit ihren Eigenschaften zusammen.

| Aufgabentyp | Beschreibung | Notation |
|---|---|---|
| Manuell | Manuelle Aufgaben sind in einem Prozessmodell zu berücksichtigen, wenn bei deren Ausführung keine Computerunterstützung möglich ist. Es handelt sich dabei beispielsweise um handwerkliche Arbeiten oder wenn Geräte bedient werden müssen. |  |
| Benutzer | Bei der Benutzeraufgabe handelt es sich um klassische workflowartige Aufgaben, bei denen der Endanwender unter Verwendung von Anwendungsprogrammen dedizierte Arbeiten durchführt. Dabei wird er aktiv mit Hilfe von Arbeitslisten oder speziellen Nachrichten über seine Prozessbeteiligung informiert. |  |
| Service | Serviceaufgaben sind zu verwenden, wenn ein beliebiger automatisierter Dienst in Anspruch genommen werden soll. Dieser Dienst kann ein Web Service oder eine andere automatisch ausführbare Anwendungsfunktionalität sein. |  |
| Empfangen | Es ist möglich, während der Prozessausführung auf das Eintreffen von Nachrichten externer Prozessteilnehmer zu warten. Wird diese Funktionalität benötigt, so ist die empfangende Aufgabe zu verwenden. Diese Aufgabe wartet so lange, bis die passende Nachricht eingegangen ist. In der Regel wird der externe Teilnehmer in Kollaborationsdiagrammen über einen Nachrichtenfluss mit der empfangenden Aufgabe verbunden. |  |
| Senden | Zur Modellierung von Aufgaben zum Versand von Nachrichten an externe Prozessteilnehmer ist die sendende Aufgabe gedacht. Sobald die Nachricht verschickt wurde, beendet sich diese Aufgabe. In der Regel wird der externe Teilnehmer in Kollaborationsdiagrammen über einen Nachrichtenfluss mit der sendenden Aufgabe verbunden. |  |
| Skript | Manche Prozessmaschinen werden mit einer eingebauten Makrosprache ausgeliefert. Diese erlaubt es Prozessmodellierern, kleinere Probleme mit Hilfe von Skripten programmatisch zu lösen. Diese Skripte laufen dann in der Prozess-Engine selbst ab. Die Skriptaufgabe deckt nun derartige Anwendungsfälle ab. Unmittelbar nach Beendigung der Skriptausführung wird auch diese Aufgabe abgeschlossen und der Prozessfluss fortgeführt. |  |

| | | |
|---|---|---|
| Geschäftsregel | Geschäftsprozesse und Geschäftsregeln ergänzen sich vortrefflich: als Vorbereitung zu Gateway-Entscheidungen können Geschäftsregeln die Entscheidungslogik kapseln und das Ergebnis dem Gateway zur Verfügung stellen. Auf diese Weise bleibt der Prozess selbst stabil und die Entscheidungsregeln können leicht änderbar ausgelagert werden. Aus diesem Grund sieht die Geschäftsregel den Aufruf von Regelmaschinen vor. |  |

***Tabelle 4: Aufgaben der BPMN 2.0 (OMG 2010a)***

### *2.1.2 Prozessablauf erläutert an einem vereinfachten Bestellprozess*

Nach dieser Kurzeinführung in die wesentlichen Aspekte der BPMN, wird im Folgenden anhand eines überschaubaren Prozesses dessen Ausführung verdeutlicht. Als Beispiel dient ein vereinfachter Bestellprozess, der in der folgenden Abbildung 3 dargestellt ist.

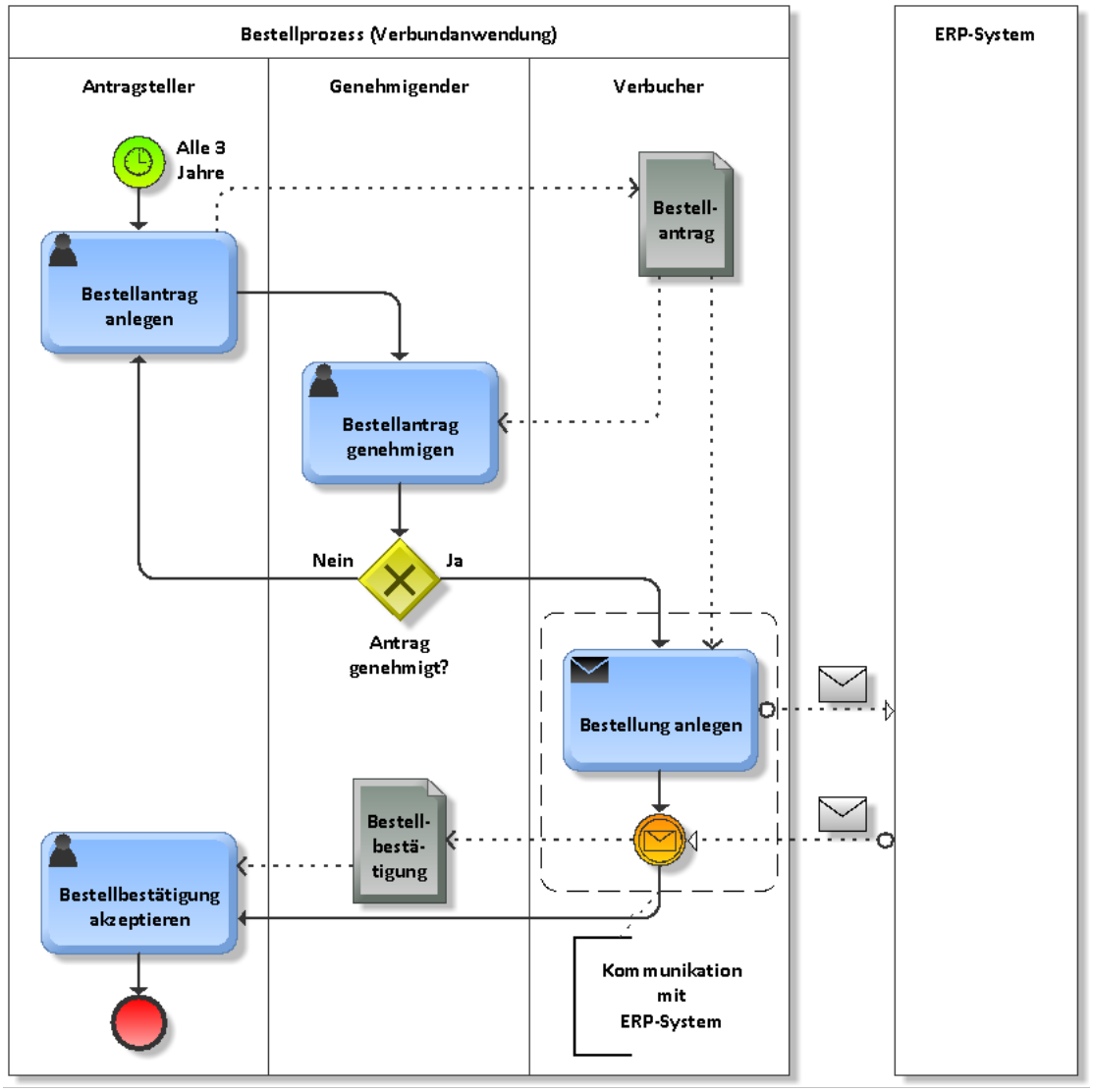



*Abbildung 3: Vereinfachter Bestellprozess modelliert mit BPMN 2.0*

Die Abbildung zeigt zwei Pools: der linke Pool repräsentiert den eigentlichen Bestellprozess, der rechte Pool ein ERP (Enterprise Resource Planning)-System, in dem die Bestelldaten hinterlegt werden. Der Bestellprozess-Pool der Verbundanwendung enthält sämtliche Prozessdetails. Er wird daher auch White Box-Pool genannt, im Gegensatz zum Pool des ERP-Systems, in dem keinerlei Sequenzflüsse eingezeichnet sind und daher auch als Black Box-Pool bezeichnet wird. Der Bestellprozess wiederum besteht aus drei Bahnen (Lanes), die die beteiligten Prozessrollen darstellen: dem Antragsteller sowie den Genehmigenden als Benutzerrollen, sowie den Verbucher als Repräsentant für einen Systemaufruf.

Der Prozess wird durch ein Timer-Startereignis gestartet. Offensichtlich ist der Prozess alle 3 Jahre automatisch zu starten, beispielsweise weil dann veraltete Laptops ausgetauscht werden sollen. Nach Start des Prozesses wird eine Markierung (Token) auf die Benutzer-Task namens *Bestellantrag anlegen* gelegt. Sie steht zur Ausführung an und entsprechend erhält der dafür zuständige Mitarbeiter eine Benachrichtigung über seine Prozessbeteiligung. Dies kann entweder in Form einer Email oder über eine dedizierte Aufgabenliste erfolgen. Der Mitarbeiter öffnet die dazugehörige Aufgabe und füllt ein Bestellformular aus. Nach abschließender Prüfung seiner Eingaben verschickt er die Bestellung und beendet damit diesen Schritt. Im Modell ist der Datenfluss ebenfalls erfasst: die getätigten Eingaben wandern in ein Datenobjekt namens *Bestellantrag*, wie der Assoziation von der Benutzer-Task zum Datenobjekt zu entnehmen ist, und stehen dort während der Laufzeit des Prozesses den anderen Schritten zur Verfügung.

Das Token wandert gemäß des modellierten Prozessflusses zur nächsten Benutzer-Task, die mit *Bestellantrag genehmigen* benannt ist. Der Schritt ist bewusst in einer separaten Bahn modelliert worden, um zum Ausdruck zu bringen, dass es sich bei Antragsteller und Genehmigenden um zwei unterschiedliche Personen handelt: es kann nicht gleichzeitig ein und dieselbe Person Antragsteller und Genehmigender sein. Folglich wird bei Erreichen dieses Schrittes eine Benachrichtigung an die nun verantwortliche Person geschickt. Sie öffnet das dazugehörige Bestätigungsformular, das aufgrund der Datenflussmodellierung offensichtlich mit den Daten des Bestellantrags gefüllt ist, und kann nun den Antrag entweder akzeptieren oder abweisen. Dazu setzt sie im Formular ein entsprechendes boolesches Feld und schließt den Schritt ab. Die Markierung verlässt daraufhin die Benutzer-Task und erreicht das Gateway. Dessen Aufgabe ist es nun, den Prozessfluss gemäß des Zustands der durch den Genehmigenden gesetzten booleschen Variable fortzusetzen. Wurde der Antrag nicht genehmigt, wandert die Markierung zurück zum *Bestellantrag anlegen*-Schritt und ermöglicht dem Antragsteller eine Korrektur. Bei einer Genehmigung hingegen wandert das Token zur sendenden Aufgabe *Bestellung anlegen*.

Die sendende Aufgabe übernimmt die Kommunikation mit dem ERP-System. Dazu verwendet sie die Daten des Bestellantrags, dargestellt durch die eingehende Assoziation vom Datenobjekt namens *Bestellantrag*, und übermittelt sie asynchron zum ERP-System. Der Nachrichtenfluss von der *Bestellantrag anlegen*-Aufgabe zum Pool

des ERP-Systems verdeutlicht diesen Vorgang. Unmittelbar nach Versand der Nachricht wandert das Token, ohne auf eine entsprechende Rückmeldung des ERP-Systems zu warten, zum nachrichtenbasierten Zwischenereignis. Erst hier stoppt der Prozess und verweilt solange, bis ihn eine wiederum asynchron eingehende Antwort erreicht. Der Nachrichtenfluss vom Black Box-Pool des ERP-Systems zum Zwischenereignis symbolisiert den Kommunikationsvorgang. Dabei werden Bestellbestätigungsdaten übertragen und in einem neuen Datenobjekt namens *Bestellbestätigung* hinterlegt. Dazu ist das nachrichtenbasierte Zwischenereignis über eine Assoziation mit dem Datenobjekt verbunden. Unmittelbar nach Empfang der Nachricht verlässt die Markierung das Zwischenereignis und wandert zur Benutzer-Task namens *Bestellbestätigung akzeptieren*.

Der Antragsteller wird aufgrund dieser Aufgabe über die erfolgreiche Abwicklung seiner Bestellung informiert. Er lässt sich die Bestellbestätigungsdaten anzeigen und beendet die Aktivität. Das Token erreicht daraufhin das Endereignis, wo es zerstört wird. Da nun keine Markierung mehr im Prozessmodell aktiv ist, kann der Prozess beendet werden.

In dem Prozessmodell wurde auch eine Gruppe verwendet, die sowohl die sendende Aufgabe als auch das nachrichtenbasierte Zwischenereignis umgibt und mit der Anmerkung *Kommunikation mit ERP-System* versehen ist. Sie verdeutlicht sehr schön, wie Gruppen zum Hervorheben zusammengehöriger Artefakte verwendet werden können. Auf die Ausführung des Prozessmodells haben Gruppen allerdings keinen Einfluss. Sie haben lediglich dokumentarischen Charakter.

Dieses Prozessmodell ist nicht vollständig, kann und soll es auch nicht sein. So gibt es beispielsweise für den Antragsteller keine Möglichkeit, aus dem Zyklus *Bestellantrag anlegen* und *Bestellantrag genehmigen* auszubrechen. Ebenso wenig hat er eine Möglichkeit, auf Fehler in der Bestellbestätigung zu reagieren. Auch die Behandlung von Kommunikationsfehlern oder eine Zeitüberwachung des Nachrichtenempfangs finden keine Berücksichtigung. Im Prinzip wurde der *Happy Path* modelliert, der im Wesentlichen die Schritte umfasst, die den Normalablauf des Prozesses aus fachlicher Sicht repräsentiert. Sämtliche Fehlerfälle wurden bewusst unterdrückt. Auf diesem Niveau werden in der Regel Prozessmodelle von den Experten der Fachabteilung erstellt. Wie noch zu sehen sein wird, erfolgt eine stetige Verfeinerung dieser

Modelle bis hin zum ausführbaren Modell. Doch zur Erläuterung wesentlicher Aspekte der BPMN sowohl hinsichtlich der Verwendung der verschieden Elemente als auch deren Bedeutung während der Ausführung genügt dieses einführende Beispiel. Im Rahmen dieser Arbeit werden weitere BPMN-Modelle folgen, die weitere Details der Notation offenbaren werden. Bereits der nächste Abschnitt verwendet die Notation zur Darstellung typischer Prozessabläufe für Verbundanwendungen.

## 2.2 Beispielprozesse für Verbundanwendungen

In diesem Abschnitt werden vier Beispiele von Prozessen diskutiert, die die vielfältigsten Einsatzszenarien von Prozessen in Verbundanwendungen dokumentieren. Sämtliche Beispiele involvieren unterschiedlichste Prozessbeteiligte und benötigen Zugriff auf die verschiedensten Dienste, die von existierenden Systemen zu erbringen sind. Es handelt sich bei den Beispielen um Prozesse…:

- …zur Anlage neuer Stammdaten (in Anlehnung an Malek & Dimitrova, 2008),
- …zur Behandlung von Problemen während der Abwicklung von Großprojekten (in Anlehnung an Stiehl & Deng, 2008),
- …zur Einsatzplanung von Schichtarbeitern (in Anlehnung an Hill & Dimitrova, 2008) sowie
- …zur Schadensmeldung im öffentlichen Bereich (in Anlehnung an SAP, 2010a).

### *2.2.1 Stammdatenbearbeitung*

Ausgangspunkt für dieses in Malek und Dimitrova 2008 diskutierte Beispiel ist der für die meisten im Internet tätigen Unternehmen typische Fall, mit neuen Kunden eine gute Geschäftsbeziehung aufbauen zu wollen und dazu deren Stammdaten benötigen. Es ist für Unternehmen heutzutage eminent wichtig, die neuen Kunden sogleich in ihr Geschäftsnetzwerk einzubinden. Schließlich soll das Vertrauen untereinander aufgebaut und die zukünftige Zusammenarbeit vereinfacht werden. Unternehmen können zusätzliches Umsatzpotenzial heben, indem sie Kundenprofile erstellen und aufgrund des Kaufverhaltens der Kunden weitere Produkte anbieten können. Der Kunde hat den Vorteil, über neueste Produkte und Serviceleistungen des Unternehmens zeitnah infor-

miert zu werden, mit einer erhöhten Kundenzufriedenheit und damit Kundenbindung als Folge. Gerade in wirtschaftlich anspruchsvollen Zeiten ist eine erhöhte Kundenbindung erfolgskritisch. Zudem ist im Internetgeschäft die Konkurrenz groß und die Hemmschwelle, den Anbieter zu wechseln, niedrig. Folglich ist die Wechselhäufigkeit relativ hoch.

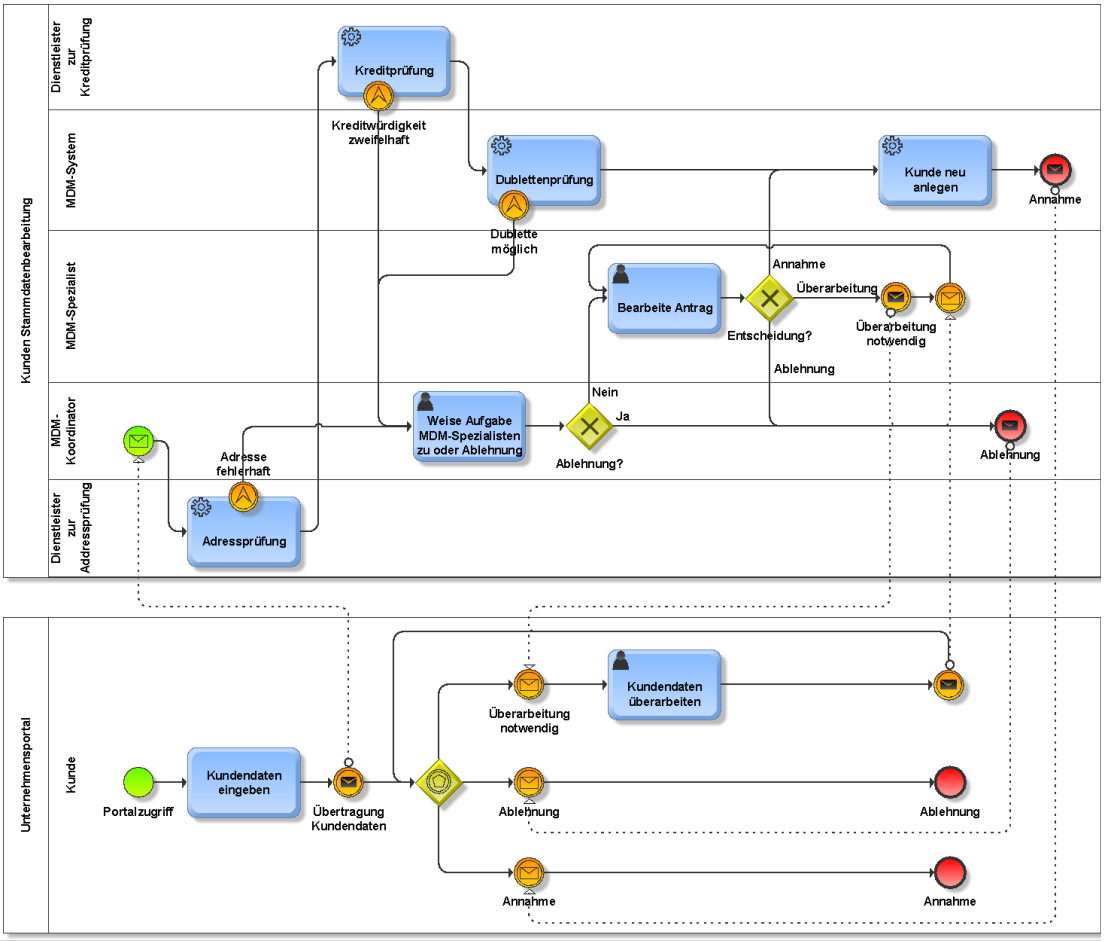



***Abbildung 4: Stammdatenbearbeitung als BPMN-Diagramm (aus Malek und Dimitrova 2008)***

Die Stammdaten spielen demnach eine entsprechende Schlüsselrolle. Doch wie kann man sie möglichst effizient erfassen? Schließlich kann nicht jeder einzelne Antrag manuell bearbeitet werden. Also muss ein neuer Prozess für eine geeignete Automatisierung sorgen. Es handelt sich dabei um einen typischen Anfrage-Bestätigungs--Prozess (vgl. dazu das Sequenzdiagramm in Abbildung 4).

Der Kunde gibt über das Firmenportal seine persönlichen Daten ein. Mit dem Versand des Eingabeformulars wird der Prozess im Unternehmen instanziiert. Zunächst

wird durch entsprechende Services, die von unterschiedlichsten Dienstleistern bereitgestellt werden können, eine Datenverifizierung durchgeführt. Klassische Plausibilitätsprüfungen wie die Überprüfung der Postleitzahl oder das Vorhandensein der Straße in der Stadt gehören genauso dazu wie die Dublettenprüfungen im vorhandenen Master Data Management-System (MDM-System). Sind die Prüfungen soweit erfolgreich abgelaufen, kommen weitere Aspekte wie die Kreditwürdigkeit hinzu. Hier gibt es ebenfalls Dienstleister, die entsprechende Services zur Verfügung stellen. Sind sämtliche Tests positiv verlaufen, kann der Kunde der Stammdatenverwaltung hinzugefügt und eine entsprechende Informationsmail an den Kunden verschickt werden. Handelt es sich allerdings um eine Dublette oder einer der Prüfungen verlief nicht zufriedenstellend, so ist eine manuelle Nachbearbeitung notwendig. In der Regel stehen dazu in den Firmen mehrere Stammdatenspezialisten zur Verfügung, die von einem Koordinator mit zu überprüfenden Kundendaten versorgt werden. Je nach Entscheidung dieser Spezialisten wird der Kunde entweder mit seinen Stammdaten aufgenommen, grundsätzlich abgelehnt oder er erhält die Möglichkeit, seine Daten nachzubessern. Wie immer die Entscheidung auch ausfällt, der Kunde wird stets über eine entsprechende Email darüber informiert.

Dieses Beispiel zeigt sehr schön die Zusammenarbeit zwischen dem Kunden und dem Unternehmen auf der einen, sowie die Kollaboration von Koordinator und den Spezialisten auf der anderen Seite. Es wird zunächst versucht, den Prozess so weit es möglich ist automatisiert abzuwickeln, ehe in Ausnahmesituationen Mitarbeiter des Unternehmens hinzugezogen werden - ein typisches Muster, das des Öfteren bei derartigen Prozessen angewandt wird. Außerdem zeigt es die Wiederverwendung existierender Funktionalitäten durch den Aufruf von Services. In der Praxis hat sich gerade dieses Szenario als eines der attraktivsten für die Unternehmen erwiesen. Es stellt somit ein klassisches Beispiel für den Einstieg der Firmen in die Welt der Verbundanwendungen dar.

### *2.2.2 Problembehandlung im Projektmanagement*

Während der Abwicklung größerer Projekte kommt es immer wieder vor, dass aufgrund von Problemen oder aufgrund von Änderungen hinsichtlich der ursprünglich angenommenen Rahmenbedingungen (wie Zielrichtung, Umfang, Design) negative Auswirkungen auf Zeit und/oder Geld zu befürchten sind. Werden derartige ungünsti-

gen Konstellationen erkannt, was ebenfalls nicht selbstverständlich ist, so stellt sich als nächstes die Frage, wie professionell die Behandlung des Problems angegangen wird. Schließlich ist ein nicht-adäquater Umgang mit solchen Situationen bzw. ein nicht-vorhandener respektive unpassender Prozess zur Begleitung der Änderungs- und Problembehandlung einer der Hauptgründe für das komplette Scheitern von Projekten. Andere Auswirkungen können ein überzogenes Projektbudget und damit ein nicht-profitabler Ausgang des Projektes oder eine endlose Verschiebung der Produktivsetzung bzw. des Verkaufsstarts eines neuen Produkts sein. Der hier vorgestellte, aus Stiehl & Deng, 2008 stammende Prozess (siehe Abbildung 5) nimmt sich genau dieser Aspekte an.

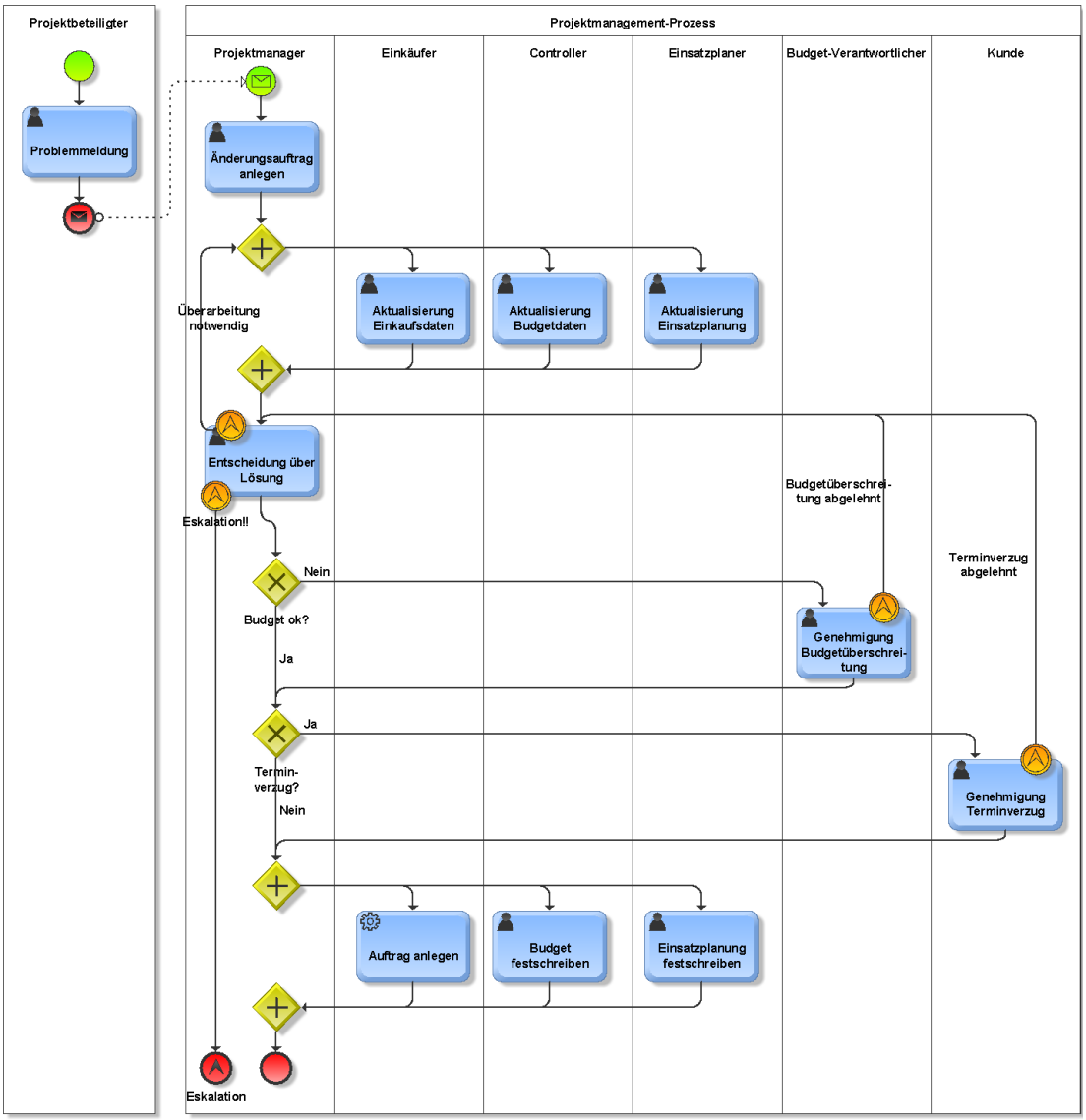



*Abbildung 5: Projektmanagement als BPMN-Diagramm (aus Stiehl & Deng 2008)*

Er erlaubt Unternehmen, auftretende Probleme frühzeitig zu erkennen, diese anschließend detailliert zu dokumentieren und dabei gleichzeitig deren Auswirkungen auf Zeit und Budget durch das Einbeziehen erfahrener Spezialisten bestmöglich zu bewerten. Diese Bewertungen beeinflussen dann wiederum den weiteren Prozessverlauf. In Summe vermeidet der Einsatz eines Problembehandlungsprozesses Situationen, in denen Veränderungen an den Rahmenbedingungen oder Störungen allgemein unentdeckt bleiben und somit den Projekterfolg gefährden. Im Wesentlichen basiert der Prozess auf den folgenden Säulen:

- Erfassung des Problems/der Änderungen in einem speziellen Formular durch beliebige Projektbeteiligte und der automatisierten Weitergabe an den Projektmanager zur Anlage eines Änderungsauftrags
- Überwachung von Status und Fortschritt der Problembehandlung
- Zuweisung von Aufgaben an die jeweiligen Experten wie Controller oder Einsatzplaner.
- Bewertung der Auswirkungen von möglichen Lösungen sowohl auf Kosten als auch auf Terminplanung.
- Einbeziehung des Kunden bei der Problemklärung
- Aktualisierung von Budget und Zeitplanung, sowie von Bestelldokumenten
- Automatisierte Einforderung von Bestätigungen seitens des Kunden bei Termin-/Kostenüberschreitungen

Der in Abbildung 5 dargestellte Prozess erhebt keinen Anspruch auf Vollständigkeit. Vielmehr soll zum Ausdruck gebracht werden, wie prinzipiell eine solche Aufgabenstellung mit Hilfe von Verbundanwendungen angegangen werden kann. Auffällig an dieser Lösung ist die stark benutzerzentrierte Ausrichtung. Nicht weniger als sieben unterschiedliche Rollen sind daran beteiligt. Der Anteil der Aktivitäten, die ohne menschliche Beteiligung abgewickelt werden kann, ist naturgemäß relativ gering, die Composite übernimmt hier primär die Rolle des Koordinators, der für eine strukturierte und damit effiziente Abwicklung sorgt.

### 2.2.3 Einsatzplanung bei Schichtarbeitern

Das dritte aus Hill & Dimitrova, 2008 stammende Beispiel beschäftigt sich mit der Einsatzplanung von Schichtarbeitern bei unvorhersehbaren Ereignissen. Das optimierte Management des Einsatzes von Schichtarbeitern ist schon seit längerem ein Problem, dessen Lösung sich Unternehmen stellen müssen. Gerade in wirtschaftlich schwierigen Zeiten suchen Firmen nach Möglichkeiten, die Schichtplanung zu optimieren und dabei gleichzeitig flexibel auf veränderte Situationen reagieren zu können. Auf der einen Seite sollen Kosten durch das Vermeiden zusätzlicher Schichten reduziert werden. Auf der anderen Seite darf es aber auch nicht dadurch zu Problemen kommen, dass zu Spitzenzeiten aufgrund fehlender Arbeiter die Produktion oder die Servicequalität gefährdet ist. Eine flexible Arbeitszeitplanung kann hier zur Lösung beitragen. Dazu eignet sich insbesondere der Einsatz der Schichtarbeit. Dieses Modell eignet sich beispielsweise für Restaurants, Hotelketten oder Produktionsbetriebe, die hochqualitative Produkte liefern bzw. Dienstleistungen zur Zufriedenheit ihrer Kunden erbringen wollen. Zur Erreichung dieser Ziele bedarf es hochmotivierter Mitarbeiter, die gleichzeitig aber auch über ihre Zeit selbst bestimmen wollen. So werden Verträge mit dem Arbeitgeber ausgehandelt, in denen die Arbeitstage sowie die Arbeitsstunden für die jeweiligen Tage festgelegt sind.

Für dieses Szenario werden zwei unterschiedliche Gruppen von Schichtarbeitern angenommen: die erste Gruppe arbeitet regelmäßig in festgelegten Schichten, während die zweite Gruppe einem Pool angehört, aus dem sich der Arbeitgeber nach Bedarf bedienen kann. Die zweite Gruppe stellt demnach einen Puffer dar, auf den das Unternehmen bei Eintreten unvorhersehbarer Ereignisse jederzeit zurückgreifen kann. Dieser Pool wird immer dann in Anspruch genommen, wenn Spitzenzeiten dies verlangen oder wenn unvorhersehbare Ausfälle nach einem schnellen Ersatz verlangen. Doch wie soll der Vorgang der Anforderung eines Ersatzes konkret ablaufen? Normalerweise übernimmt ein Einsatzplaner eine solche Aufgabe, sobald derartige Situationen bekannt werden. Allerdings ist es unrealistisch anzunehmen, dass Ausfälle aufgrund von (beispielsweise) Krankheit immer während der Arbeitszeit des Einsatzplaners gemeldet werden. Abwesenheitsmeldungen können jederzeit eintreffen, auch außerhalb der normalen Arbeitszeiten. Es soll dem Mitarbeiter vielmehr ermöglicht werden, seinen Arbeitgeber stets dann zu informieren, wenn eine Abwesenheit abzusehen ist, also insbesondere in den frühen Morgenstunden, am Abend oder an den Wochen-

enden. Problematisch wird es auch, wenn der Einsatzplaner für eine Vielzahl von Schichtarbeitern verantwortlich ist. Es ist ausgeschlossen anzunehmen, er allein könnte die komplette Einsatzplanung überblicken und managen. Gelöst werden kann dieses Problem nur mittels eines nahezu vollständig automatisierten Ablaufs der Einsatzplanung. Eine mögliche Implementierung ist in Abbildung 6 dargestellt.

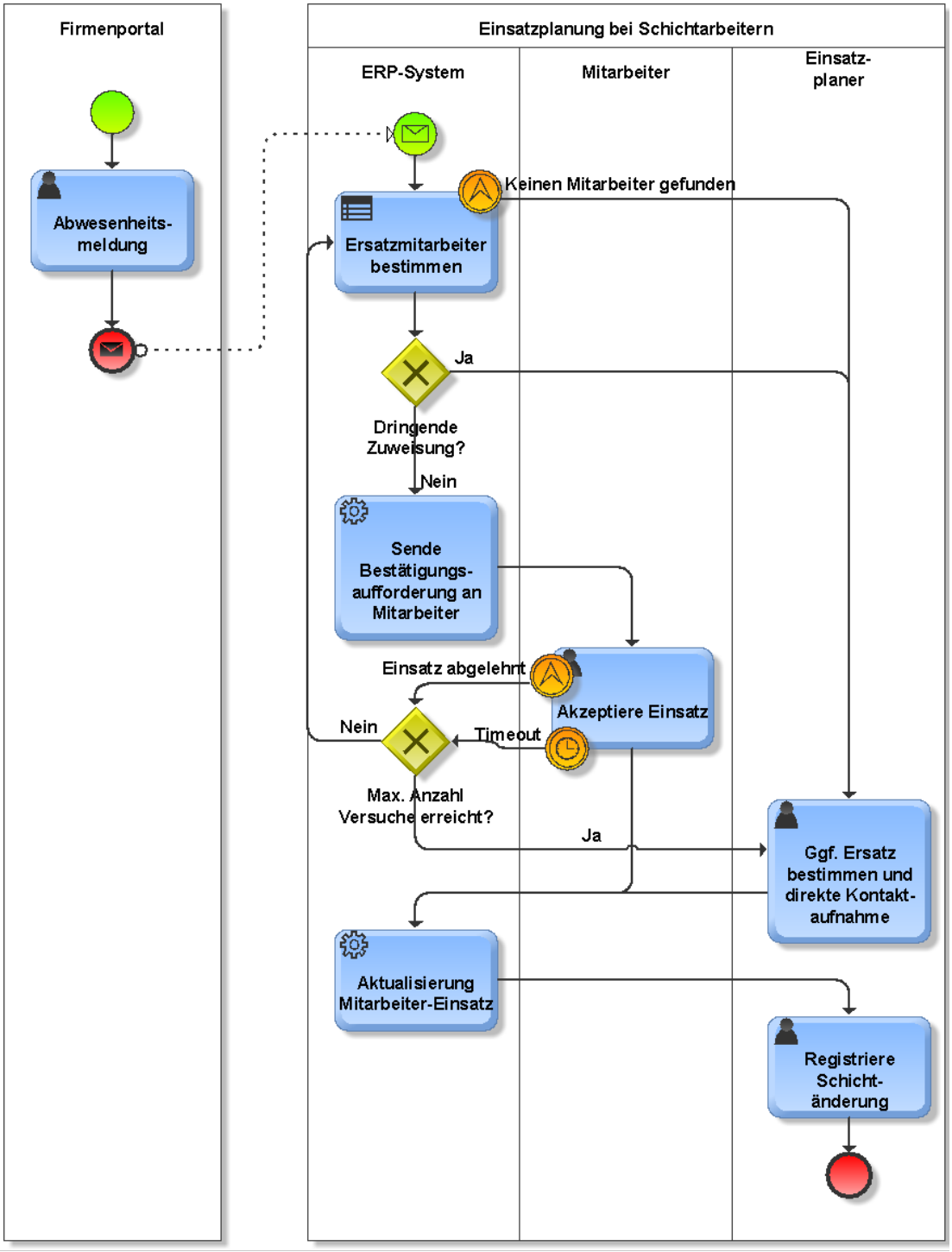



*Abbildung 6: Schichtarbeitereinsatz als BPMN-Diagramm (aus Hill & Dimitrova, 2008)*

Zur Verdeutlichung des Prozessablaufs sei für dieses Beispiel eine Abwesenheitsmeldung angenommen. Diese wird vom betroffenen Mitarbeiter über das Firmenportal eingegeben und abgeschickt. Dadurch wird der eigentliche Prozess zur Schicht-Einsatzplanung angestoßen. Automatisch wird aufgrund von Regeln nach einem adäquaten Ersatz gesucht. Ist dieser gefunden, erhält der betroffene Mitarbeiter automatisch eine Benachrichtigung, beispielsweise über SMS oder über spezielle Meldegeräte wie sie auch bei Ärzten, Feuerwehren oder Polizisten gebräuchlich sind. Nun sind verschiedenen Reaktionen des derart kontaktierten Mitarbeiters denkbar

- Er übernimmt die Schicht: das Problem ist gelöst, der Einsatz kann entsprechend verbucht und der Einsatzplaner über die Änderung informiert werden, die dieser dann an seinem nächsten Arbeitstag einsehen kann.
- Er lehnt die Schicht ab: der Vorgang der automatischen Zuweisung wird wiederholt
- Er reagiert nicht: dann muss nach Ablauf einer gewissen Wartezeit erneut die automatische Zuweisung aktiviert werden.

Nur wenn die automatische Zuweisung keine Alternative finden kann, wenn eine besonders kritische Situation eintritt oder wenn eine konfigurierbare maximale Anzahl an Alternativen erfolglos war, wird der Einsatzplaner explizit in den Prozess involviert. Dieses Beispiel zeigt einmal mehr, wie eine hochgradige Automatisierung zur Effizienzsteigerung in Unternehmen beitragen kann. Auch das Pattern zur Einbeziehung von Personal nur in wirklichen Ausnahmesituationen tritt hier wieder auf.

### *2.2.4 Schadensmeldung im öffentlichen Bereich*

Das letzte in SAP 2010a diskutierte Beispiel gehört in den Bereich der Selbstbedienungsszenarien im öffentlichen Bereich. Gerade hier lassen sich viele Automatisierungsszenarien umsetzen. In dem hier behandelten Schadensmeldungsszenario werden gleich mehrere Missstände adressiert. In der Regel gibt es keine etablierten Prozesse in der öffentlichen Verwaltung, denen bei unerwarteten Wartungsarbeiten (z.B. Baum, der sich auf einer Straße quergelegt hat; Defekte an öffentlichen Einrichtungen) gefolgt werden könnte. Vielmehr wird vom internen Personal ad hoc entschieden, wie weiter zu verfahren ist. Bei einer steigenden Zahl solcher außergewöhnlichen

Ereignisse kann es leicht zur Überlastung kommen. Durch eine fehlende Automatisierung sind die Gesamtkosten zur Beseitigung des Problems entsprechend hoch. Neben diesen prozessrelevanten Aspekten ist die Unzufriedenheit bei dem Bürger selbst ein weiterer wichtiger Punkt, da es ihm erschwert wird, mit den richtigen Stellen in Kontakt zu treten, um für eine schnellstmögliche Abwicklung derartiger Probleme zu sorgen.

Von daher ist es das zentrale Ziel dieses Beispiels, für klar definierte Prozesse zu sorgen und damit die o.a. Schwachstellen zu adressieren. Der Bürger wird besser am Prozess beteiligt, Kosteneinsparpotenzial wird durch die Prozessautomatisierung gehoben, die Qualität wird durch die Beauftragung externer Spezialfirmen gesteigert und das Personal kann sich wieder auf die eigentlichen Aufgaben, wie die Überwachung und Durchführung von standardmäßigen Instandhaltungsarbeiten, konzentrieren. Abbildung 7 fasst die wesentlichen Schritte des Prozesses zusammen.

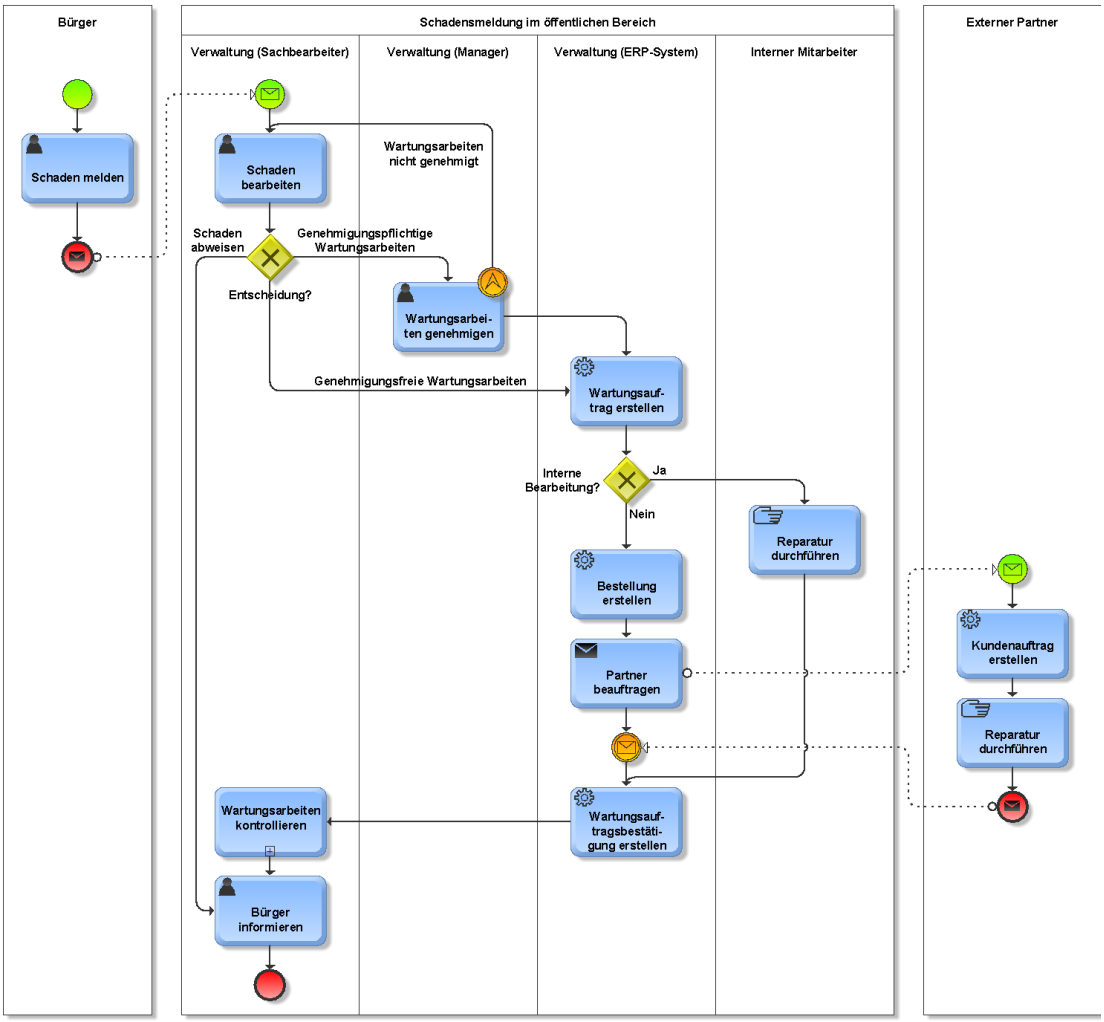



*Abbildung 7: Schadensmeldung im öffentlichen Bereich als BPMN-Diagramm (aus SAP, 2010a)*

Er wird durch eine Schadensmeldung des Bürgers initiiert. Er kann diese Meldung durch ein modernes, leicht zugängliches Formular, das im Internet als Teil eines Bürgerportals vorliegt, abgeben. Nach Übermittlung des Schadens (beispielsweise eine defekte Wippe auf einem Kinderspielplatz, einschließlich Foto des Defekts) wird ein Sachbearbeiter der Instandhaltungsabteilung automatisch über den Mangel informiert. Nach Prüfung der Schadensmeldung können anschließend automatisierte Regeln darüber entscheiden, ob eine Genehmigung durch einen Vorgesetzten notwendig ist, beispielsweise wenn die zu erwartenden Reparaturkosten einen bestimmten Betrag übersteigen, oder ob diese unterbleiben kann. Ebenfalls automatisch kann über Regeln gesteuert werden, ob die Reparatur durch das eigene Personal durchgeführt werden kann oder ob dafür eine externe Spezialfirma beauftragt werden muss. Je nachdem, wie durch die Regeln der weitere Prozessablauf beeinflusst wird, sind zusätzliche Prozessbeteiligte zu informieren. So kann beispielsweise nach Genehmigung durch den Vorgesetzten des Sachbearbeiters ein externes Gartenbauunternehmen mit der Behebung des Schadens durch eine Business-to-Business-Integration beauftragt werden. Nach erfolgreicher Reparatur wird wiederum automatisiert die Rechnung an die betroffene Stadt gesendet. Letztendlich hat dann der Sachbearbeiter die Reparaturleistung zu prüfen und schließlich die Zahlung der Rechnung durch seine Genehmigung zu veranlassen. Im letzten Schritt benachrichtigt der Sachbearbeiter den Bürger, der den Prozess ursprünglich startete, über die erfolgreiche Schadensbehebung.

Dieses Beispiel verdeutlicht einmal mehr den integrativen Teil von Verbundanwendungen: nicht nur, dass sich das Szenario über verschiedenste Rollen unterschiedlichster Abteilungen innerhalb der Behörde erstreckt – es werden auch explizit externe Partner/Lieferanten in den Prozessablauf involviert. Soll eine solche Lösung als Produkt auch in anderen Städten zum Einsatz kommen, wird die Bedeutung einer nachhaltigen Architektur für Composite Applications umso deutlicher. Von daher wird dieses Thema im nachfolgenden Kapitel im Detail betrachtet.

# 3 Architektur von Verbundanwendungen

Aufgrund der in Kapitel 2 besprochenen Eigenschaften von Verbundanwendungen stellt sich zwangsläufig die Frage, wie diese nun konkret umgesetzt werden können. Dieser Frage widmet sich dieses Kapitel. Dabei wird zunächst kurz die Vorgehensweise bei der Entwicklung von Verbundanwendungen angesprochen: womit soll begonnen werden? Sind die neu zu implementierenden Geschäftsprozesse der Ausgangspunkt aller Überlegungen und leiten sich anschließend daraus durch Verfeinerungen die Anforderungen an eine Composite ab? Oder ist mit den in der Systemlandschaft bereits existierenden Diensten zu beginnen, um sie anschließend zu neuen Prozessen zusammenzufügen? Wie zu sehen sein wird, müssen für Verbundanwendungen bestimmte Besonderheiten, wie die vollständige Unabhängigkeit von den Backend-Systemen, berücksichtigt werden, die entscheidenden Einfluss auf das Vorgehen haben. Mit der vorgestellten Methodik sind anschließend die wesentlichen Bestandteile der Composite zu ermitteln und entsprechend einer Spezifikation zusammenzutragen. Dabei wird sich herausstellen, dass es für Verbundanwendungen sinnvoll ist, zwischen lose-gekoppelten betriebswirtschaftlichen und technischen Prozessen zu unterscheiden und diese auch streng separiert zu implementieren. Welche Rolle die „Lose Kopplung“ spielt und welche Funktionalitäten sich schließlich in welcher Schicht wiederfinden, wird ebenfalls zu diskutieren sein.

Abgeschlossen wird dieses Kapitel mit einer Einschätzung über die Bedeutung von Repositories im Zusammenhang mit Composite Applications.

## 3.1 Methodisches Vorgehen: Top-Down-Ansatz

Eine der Kernfragen bei der Entwicklung von Verbundanwendungen ist sicherlich, wie die Herangehensweise für derartige Lösungen wohl aussieht. Betrachtet man hierzu Empfehlungen in der Literatur, wobei sich aufgrund der Ähnlichkeiten zu SOA insbesondere Quellen aus diesem Gebiet als Einstieg eignen, so wird zu einer Kombination von Top-down und Bottom-up geraten (Josuttis 2008 S. 106, Bloomberg & Schmelzer 2006). Dabei wird unter einer Top-down-Vorgehensweise die Verfeinerung eines Problems durch Zerlegung in immer kleinere Einheiten verstanden, bis sich schließlich die Kernbestandteile der Anwendung wie beispielsweise Geschäftsobjekte und Basisdienste als Ergebnis ergeben (Dekomposition).

Im umgekehrten Fall, dem Bottom-Up-Ansatz, werden ausgehend von den existierenden Diensten der zu integrierenden Anwendungen neue Prozesse erstellt (Komposition). Interessant ist in diesem Zusammenhang das folgende Zitat aus Bloomberg & Schmelzer 2006, das auch in Josuttis 2008, S. 106 zu finden ist:

„*Der Ansatz in SOA sollte sowohl top-down (mit Prozess-Dekomposition) als auch bottom-up (existierende Funktionalitäten zu Services machen und diese zu Prozessen komponieren) sein. Verwendet man nur den Top-down-Ansatz, ist es recht wahrscheinlich, dass man die Entwicklung von Services empfiehlt, die aus technischer Sicht schwierig oder kompliziert zu implementieren sind. Nutzt man nur den Bottom-up-Ansatz, führt das leicht zu Services, die man gar nicht benötigt oder (schlimmer noch) die überhaupt nicht die Anforderungen des Business erfüllen.*“

Wie sind solche Aussagen im Kontext von Verbundanwendungen zu bewerten? Bereits in Kapitel 2 wurde detailliert beschrieben, dass eine herausragende Eigenschaft von Composite Applications deren konsequente Unabhängigkeit von den Backend-Systemen darstellt. Von daher scheidet der Bottom-up-Ansatz von vornherein aus. Im Gegenteil: Architekten von Composites ist zu empfehlen, gänzlich ohne Kenntnis der IT-Landschaft die Konzeption einer Verbundanwendung anzugehen: dies vermeidet von Anfang an eine zu große Abhängigkeit von Anwendungsfunktionalitäten, deren Schnittstellen und den darin verwendeten Datentypen. Stattdessen muss eine Composite ihren Startpunkt grundsätzlich bei den neu zu implementierenden Prozessen finden. Die zu implementierende Fachlichkeit ist der Treiber einer jeden Verbundanwendung. Dabei stehen die betriebswirtschaftlichen Prozesse im Mittelpunkt: ausgehend von der Sequenzfolge der einzelnen Prozessaktivitäten erschließen sich die in Abbildung 2 dargestellten Komponenten einer Composite:

- Prozesse
- Benutzeroberflächen
- Geschäftsobjekte
- Dienste

Grundsätzlich sind sämtliche Komponenten aus der Sicht der konkret umzusetzenden Composite zu definieren. Die zu realisierenden Prozesse verfolgen ausschließlich das Ziel, ein konkretes fachliches Problem zu lösen. Ohne Kompromisse und Rücksichtnahme auf existierende Landschaften und Funktionen sind die Prozessabläufe so zu gestalten, dass für das Unternehmen eine optimale Lösung entsteht. In den Prozessen sind die Prozessrollen zu identifizieren und ausgehend von den Prozessrollen die dazugehörigen Benutzeroberflächen zu finden, die die jeweilige Rolle in ihrer jeweiligen Funktion im Prozess bestmöglich unterstützt. „Bestmöglich unterstützt“ bedeutet hier eine Reduzierung der Felder auf den Bildschirmmasken auf das unbedingt notwendige Maß. Es sind die Informationen anzuzeigen bzw. die Funktionalitäten wie Suchen und Wertehilfen zur Verfügung zu stellen, die ein effizientes Arbeiten ermöglichen. Dies deckt sowohl die Unterstützung bei der Eingabe von Daten durch passende Hilfen als auch die Anzeige von Charts und Diagrammen zur Entscheidungsfindung ab.

Weiterhin lässt sich aufgrund der Prozesse detailliert ableiten, auf welchen Daten die Composite operiert. Die einzelnen Attribute/Felder lassen sich dann wiederum zu Geschäftsobjekten zusammenfassen. Auch hier gilt: Reduktion auf das unbedingt Notwendige. Auch wenn in den Systemen der vorhandenen IT-Landschaft mit Sicherheit komplexe Strukturen für Geschäftsobjekte wie Kunde, Produkt oder Auftrag mit einer Vielzahl von Attributen existieren, so werden Verbundanwendungen in der Regel nur auf einen sehr geringen Bruchteil dieser Daten arbeiten. Von daher macht es keinen Sinn, sich diese Komplexität in die Composite zu holen. Die Beschränkung auf das Wesentliche hat höchste Priorität.

Die benötigten Dienste sind ebenfalls ausschließlich aus betriebswirtschaftlicher Sicht zu definieren. Unabhängig davon, ob ein Dienst innerhalb einer Benutzeroberfläche benötigt wird oder aus einem Prozessschritt heraus gerufen werden soll: einzig der betriebswirtschaftliche Nutzen, die dieser Service zu erbringen hat, steht im Mittelpunkt. In diesem Punkt unterscheiden sich Verbundanwendungen grundlegend von herkömmlichen SOA-basierten Anwendungen: bei SOA-basierten Anwendungen werden existierende Dienste in der Regel direkt aus der Endanwendung heraus aufgerufen. Bei Composite Applications ist das niemals der Fall. Für Verbundanwendungen werden grundsätzlich sämtliche nach außen abgehende Aufrufe neu definiert. Die Schnittstellen werden dabei wiederum ausschließlich nach den betriebswirtschaftli-

chen Bedürfnissen der Composite ausgerichtet, d.h. sowohl Eingabe- als auch Ausgabeparameter orientieren sich an den Prozessen der Composite. Werden beispielsweise in einem neuen Prozess zur Auftragsanlage lediglich die Produktnummer und die dazugehörige Menge benötigt, so besitzt die öffentliche Schnittstelle dieses Dienstes zur Erzeugung eines Auftrags genau diese beiden Daten als Eingabeparameter. Niemals wird eine Composite eine existierende Schnittstelle importieren und darauf aufbauend die Aufrufe in Prozessschritte oder Benutzeroberflächen einsetzen. Die benötigte Funktionalität wird stattdessen durch eigens zu definierende Schnittstellen abstrahiert. Die in dem eingangs dieses Abschnitts zitierte Gefahr, aufgrund des Top-down-Ansatzes die Entwicklung von Services zu empfehlen, die aus technischer Sicht schwierig oder kompliziert zu implementieren seien, hat sich in der Praxis nicht in dem Maße bestätigt, wie dies ursprünglich befürchtet wurde. Durch lose Kopplung und asynchroner Kommunikation lässt sich insbesondere bei verändernden Services (schreibende, löschende, aktualisierende Dienste) eine zeitliche Entkopplung zwischen Aufrufer und Aufgerufenen erreichen, die eine vernünftige Umsetzung auch von komplizierten Funktionalitäten erlaubt. Auf den Aspekt der losen Kopplung wird in einem dedizierten Abschnitt noch im Detail eingegangen.

Die Details von Prozessen, Benutzeroberflächen, Geschäftsobjekten und Diensten lassen sich dabei nur in enger Zusammenarbeit von Fach- und IT-Abteilung finden. Welche Informationen in diesen Zusammenhang eine Rolle spielen, soll der folgende Abschnitt beleuchten.

## 3.2 Spezifikation von Verbundanwendungen

Sowohl in Rauscher & Stiehl 2008 als auch in Stiehl 2006 wird ausführlich auf die Spezifikation von Verbundanwendungen eingegangen. Allerdings wird in den Dokumenten noch davon ausgegangen, dass verfügbare Dienste direkt aus der Composite heraus angesprochen werden. Diese Vorgehensweise ist aufgrund der Erläuterungen des vorangegangenen Abschnitts nun nicht mehr gängige Praxis, da sich zum einen Abhängigkeiten ergeben, die es unbedingt zu vermeiden gilt, zum anderen die Komplexität aufgrund der dadurch eingehandelten Probleme noch zunimmt. Dennoch finden sich in den Papieren wichtige Ansätze, die hier kurz zusammengefasst wiedergegeben werden, da sie für die weitere Diskussion von Verbundanwendungen von Be-

deutung sind. Im Wesentlichen stellt dieser Abschnitt eine Übersicht zusammen, welche Daten/Anforderungen konkret zur Umsetzung von Composite Applications zusammengetragen werden müssen, damit eine anschließende Implementierung möglich ist. Erbracht werden muss diese Leistung von den Experten der Fachabteilungen, ggf. unterstützt durch Kollegen der IT-Abteilung. Da eine Composite fachlich motiviert ist, sind auch nur die Fachexperten in der Lage, den größten Teil der benötigten Informationen bereitzustellen. Die nachfolgenden Abschnitte sind als eine Art Checkliste zu verstehen, an denen sich der Ersteller der Spezifikation orientieren kann. Die hier aufgeführte Auflistung konzentriert sich allein auf Composite-relevante Aspekte und erhebt keinen Anspruch, die zu erstellende Software vollständig zu spezifizieren. Sämtliche Aspekte werden dabei verbal beschrieben.

Wie die einzelnen Punkte auch durch Verwendung geeigneter Modelle dargestellt werden können, zeigt Pitschke 2010 in seinen Ausführungen über die *Unternehmensmodellierung für die Praxis*. Selbst für die verbale Beschreibung der Anforderungen sind sogenannte Anforderungsschablonen denkbar. Dabei handelt es sich um Satzmuster, die den Spielraum für fehlerhafte Interpretationen minimieren sollen, da Fehlinterpretationen oft Risiken für Projekte darstellen. Pitschke führt eine aus Rupp 2009 stammende Schablone als Beispiel auf, um diesen Sachverhalt zu verdeutlichen. Abbildung 8 veranschaulicht eine solche Schablone.

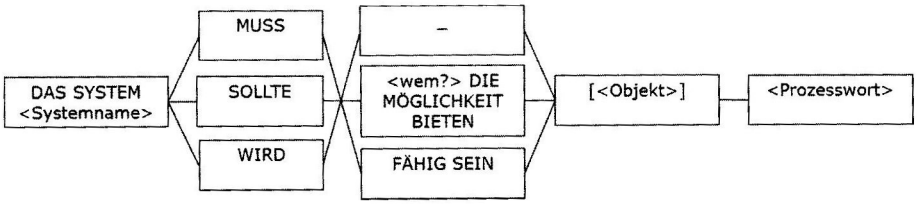



***Abbildung 8: Beispiel für eine Anforderungsschablone (aus Pitschke 2010, ursprünglich aus Rupp 2009)***

Es ist zu empfehlen, sich auch bei der Spezifikation von Verbundanwendungen an derartige Muster zu halten, um eben den Auslegungsspielraum einzuengen und somit zu eindeutigeren Beschreibungen zu kommen. Dabei sollten die einzelnen Aspekte zwecks besserer Erfassung und Übersichtlichkeit in tabellarischer Form zusammengetragen werden (Pitschke 2010, Stiehl 2006). Für Verbundanwendungen liefert Stiehl

2006 bereits einen umfassenden Satz von Tabellen, die die weiter unten folgenden Spezifikationsaspekte optimal unterstützen.

Ergänzt werden könnten die informellen Erläuterungen zudem um visuelle Darstellungen. Pitschke empfiehlt hier beispielsweise die OMG Systems Modeling Language (SysML 2010) zur Darstellung sowohl der Zusammenhänge zwischen einzelnen Anforderungen, als auch zur Verknüpfung von Anforderungen mit anderen Modellelementen wie z.B. Testfällen, die der Verifizierung der Anforderungen, denen sie zugeordnet wurden, dienen. Auf diese Weise werden Querbezüge und Abhängigkeiten offensichtlich, die in rein verbalen Ausführungen verloren gehen.

Die Vorgehensweise zur Spezifikation von Verbundanwendungen folgt dabei dem Top-down-Ansatz, der im vorangegangenen Abschnitt bereits skizziert wurde. Als wesentliche Schritte ergeben sich demnach:

1. Allgemeine Informationen über die Composite Application
2. Prozessinformationen
3. Ausnahmebehandlung
4. Geschäftsobjekte
5. Benutzeroberflächen
6. Dienste

### *3.2.1 Allgemeine Informationen über die Composite Application*

Zunächst muss sich die Fachabteilung über das Ziel der Verbundanwendung im Klaren sein. In der Regel sollen Composites genau dann eingesetzt werden, wenn existierende Schwachpunkte beseitigt, Wettbewerbsvorteile weiter ausgebaut oder neue Geschäftsfelder erschlossen werden sollen. Dabei stehen diese Ziele grundsätzlich im Einklang mit der von der Unternehmensleitung vorgegebenen strategischen Ausrichtung (Alignment-Aspekt - Banerjee 2006). Das Management muss nachvollziehen können, welches Problem konkret mit der neuen Lösung adressiert werden soll. Belegen lässt sich das Problem idealerweise durch Auswertungen über den aktuellen Zustand des nicht vorhandenen, nicht effizienten oder fehlerhaft abgewickelten Prozesses. Mögliche Defizite lassen sich beispielsweise durch folgende Zahlen belegen:

- Zeitaspekte
    - Lange Durchlaufzeiten
    - Lange Bearbeitungszeiten
    - Ungünstiges Verhältnis von Durchlauf- zu Bearbeitungszeiten
    - Unstimmiges Verhältnis zwischen rechtzeitigen Lieferungen im Vergleich zu verspäteten Lieferungen
- Kostenaspekte
    - Fehlerbearbeitungskosten
    - Prüfkosten
    - Prozesskosten
- Qualitätsaspekte
    - Niedriger First-Pass-Yields-Anteil
    - Erhöhte Fehlerrate
    - Erhöhte Rücklaufrate aufgrund von Qualitätsmängeln
    - Verschwendung von Material

Darauf aufbauend kann nun die Lösungsidee präsentiert werden: wie können die identifizierten Mängel aufgrund einer neuen Lösung adressiert und nachhaltig beseitigt werden? Unter Umständen können Prozesssimulationen bereits zu diesem sehr frühen Stadium die Argumentation für die Verbundanwendung unterstützen. Kosten/Nutzen-Analysen dürfen in dieser Phase nicht fehlen. Das Management muss von der Idee überzeugt sein. Daraus resultierende positive Aspekte sind beispielsweise Effizienzsteigerungen, Flexibilitäts- und/oder Qualitätsverbesserungen, Kostenreduktion, reduzierte Fehlerraten, Durchsatzerhöhung oder Steigerung der Kundenzufriedenheit. Wie auch immer die Verbesserungen erzielt werden können, sie müssen auch monetär belegbar sein. Nur dann wird das Management einem Start des Projektes zustimmen.

Eine umfassende Dokumentation dieser Aspekte ist auch im späteren Projektverlauf wichtig. So lässt sich jederzeit kontrollieren, ob die ursprünglich angestrebten Ziele auch tatsächlich erreicht wurden. Ebenso können anstehende Entscheidungen stets gegenüber den ursprünglich ausgegebenen Zielen reflektiert und entsprechend

gehandelt werden. Zusammenfassend stellen diese Informationen die Daseinsberechtigung der Verbundanwendung dar und sind somit essenziell.

### *3.2.2 Prozessinformationen*

In diesem Teil der Spezifikation sind sämtliche in der Verbundanwendung beteiligten Prozesse aufzulisten, ihre Funktion(en) kurz zu beschreiben sowie deren Beitrag zur Zielerreichung der Composite darzustellen. Anschließend ist jeder einzelne Prozess detailliert zu charakterisieren, wobei im Wesentlichen folgende Informationen *pro Prozess* zusammenzutragen sind:

- Allgemeine Prozessinformationen
- Beteiligte Prozessrollen
- Visualisierung des Prozessflusses
- Detailinformationen zu den Prozessschritten
- Beschreibung des Datenflusses innerhalb des Prozesses (Prozesskontext)

Im Folgenden werden die einzelnen Punkte kurz beschrieben und Anhaltspunkte gegeben, an denen sich während der Erstellung einer Spezifikation orientiert werden kann.

#### 3.2.2.1 Allgemeine Prozessinformationen

Mit „Allgemeine Prozessinformationen“ sind primär die Startbedingungen des Prozesses sowie dessen zeitliche Rahmenbedingungen gemeint. Typische Fragestellungen, die hier zu berücksichtigen sind, lassen sich wie folgt zusammenfassen:

- Unter welchen Bedingungen ist der Prozess zu starten?
    - Zeitlicher Aspekt
        - Start zu einem bestimmten Zeitpunkt (z.B. einmalig am 24.12. um 14:00 Uhr oder regelmäßig jeden Mittwoch um 6:00 Uhr)
        - Start nach einer bestimmten Zeitspanne (z.B. einmalig in genau 7 Stunden oder regelmäßig alle 6 Stunden)
        - Start in Relation zu einem anderen Zeitpunkt (z.B. 1 Monat vor dem 24.12.)
    - Bedingungen

        - Start des Prozesses, wenn eine oder mehrere Bedingungen erfüllt sind, wie beispielsweise das Auffüllen eines Lagers mit einem bestimmten Produkt, sobald ein vorgegebener Schwellenwert unterschritten wurde
    - Nachrichten
        - Der Prozess wartet auf das Eintreffen einer speziellen Nachricht.
    - Ereignisse
        - Insbesondere in Ausnahmesituationen spielen Ereignisse eine entscheidende Rolle: sie werden immer dann geworfen, wenn außergewöhnliche Situationen eingetreten sind, die einer besonderen Behandlung bedürfen, z.B. bei Lieferverzug, Maschinenausfall, Unfall oder ähnlichen Umständen.

  Selbstverständlich kann der Start auch von mehreren der oben genannten Bedingungen abhängen. Dann ist dies entsprechend zu vermerken.

- Sind zeitliche Randbedingungen für den gesamten Prozess zu beachten?
  Muss der Prozess innerhalb einer bestimmten Zeitspanne abgeschlossen werden und falls ja, wie ist bei Überschreitung zu reagieren? Wer ist gegebenenfalls zu informieren und welche Folgeaktivitäten sind zu initiieren?

- Allgemeine Prozessrollen
  Wer darf die Ausführung des Prozesses überwachen und dadurch Einblick in den Ausführungszustand nehmen? Wer darf Aktivitäten, die beispielsweise aufgrund einer Urlaubssituation nicht abgearbeitet werden können, an einen Vertreter weiterleiten? Wer darf selbst eine unbearbeitete Aktivität selbstständig übernehmen? Und wer darf auf keinen Fall Einblick nehmen, Aktivitäten verteilen oder selbst bearbeiten? Durch diese Fragen wird nach Antworten für die typischen Prozessrollen wie Administrator, potenzielle Aufgabenbesitzer bzw. ausgeschlossene Aufgabenbesitzer gesucht.

#### 3.2.2.2 Beteiligte Prozessrollen

Mit den beteiligten Prozessrollen wird festgelegt, wer während der Prozessdurchführung konkret welche Aktivitäten zu übernehmen hat. Es genügt eine einfache Liste der Prozessrollen und eine Beschreibung, welche Funktion die Rolle innerhalb des Prozesses zu übernehmen hat. Diesen Rollen werden zur Laufzeit konkrete Benutzer und Gruppen aus den in den Unternehmen eingesetzten Benutzerverwaltungen zugeordnet. Diese Aufgabe befriedigend zu lösen ist alles andere als trivial. Allerdings wird im Rahmen dieser Arbeit nicht weiter auf die Komplexität bei der Zuweisung von Benutzern zu Prozessrollen im Rahmen von Workflow-Managementsystemen bei gleichzeitiger Berücksichtigung der Unternehmensorganisation eingegangen. Einen guten Überblick zu diesem Thema ist in Zur Mühlen (2004) zu finden, einschließlich einer Übersicht weiterführender Arbeiten.

#### 3.2.2.3 Visualisierung des Prozessflusses

Die Visualisierung des Prozessflusses ist von fundamentaler Bedeutung: es dient als Kommunikationsmittel zwischen dem Experten der Fachabteilung und dem Architekten, der die Prozessanforderungen umzusetzen hat. Zur Prozessmodellierung hat sich mittlerweile der OMG-Standard BPMN (Business Process Model and Notation: OMG 2010a) als beste Alternative herausgestellt. Sie wird sowohl von der Fachseite als auch von der IT-Seite akzeptiert und stellt somit eine Brücke zwischen diesen Welten dar (Stiehl 2009a-2009d). Letztendlich soll das BPMN-Prozessmodell die Frage beantworten, was wann unter welcher Bedingung durchgeführt werden soll. Allerdings sollten in der Spezifikation nicht sämtliche Details ausmodelliert werden, da dies nicht Aufgabe einer Fachabteilung sein kann. Prinzipiell sollte der Fachspezialist zumindest folgende Informationen bereitstellen:

1. Den Sequenzfluss der Prozessschritte für den sogenannten „Happy Path“, also der normale Prozessfluss, der sich ergibt, wenn keinerlei Fehler auftreten. In der BPMN wird dies über Sequenzdiagramme realisiert.
2. Typisierung der Aktivitäten, ob diese also entweder vollautomatisiert oder unter Einbeziehung eines Endbenutzers abgewickelt werden. In der BPMN haben sich dafür Benutzer- bzw. Service-Aufgaben etabliert.

3. Zuweisung von Rollen zu den Benutzer-Aufgaben: hierfür sind in der BPMN Pools und Lanes vorgesehen.
4. Behandlung fachlicher Ausnahmesituationen. Hierfür eignen sich aus der BPMN-Palette entweder Ereignisse oder unterschiedliche Endzustände, die über Gateways angesprochen werden. Es ist zu beachten, dass in den Modellen ausschließlich die fachlichen Ausnahmen wie Verzögerungen, Qualitätseinbußen oder Kostenüberschreitungen zu berücksichtigen sind. Technische Ausnahmen wie Verbindungsprobleme, Datenbankausfälle oder falsche Datenformate sind aus fachlicher Sicht vorerst vernachlässigbar. Das BPMN-Modell bleibt dadurch überschaubar.

Sowohl Freund, Rücker & Henninger (2010) als auch Silver (2009) empfehlen eine Reduzierung der auf der Fachseite zu verwendenden BPMN-Artefakte auf ein absolutes Minimum. So empfiehlt Silver beispielsweise die Verwendung folgender BPMN-Konstrukte für die Fachabteilung (Silver 2009, S. 25):

- Pools und Lanes
- Benutzer- und Service-Aufgabe
- Unterprozesse (sowohl zu- als auch aufgeklappt)
- Startereignis (untypisiert, Zeitereignis, Nachrichtenereignis)
- Endereignis (untypisiert, Nachrichtenereignis, Terminierungsereignis)
- Exklusives und paralleles Gateway
- Sequenzfluss und Nachrichtenfluss
- Datenobjekte, Datenspeicher und Nachrichten
- Anmerkungen (Kommentare)
- Link-Ereignisse zur Fortsetzung von Prozessmodellen auf anderen Seiten

Zudem empfiehlt er für die Modellierung von Geschäftsprozessen ein stufenweises Vorgehen (siehe auch Silver 2008 und Silver 2009, S. 7/8):

- Stufe 1: Beschreibende Modellierung (Descriptive Modeling)
- Stufe 2: Analytische Modellierung (Analytical Modeling)

- Stufe 3: Ausführbare Modellierung (Executable Modeling)

Für jede dieser Stufen wird eine bestimmte Menge von BPMN-Konstrukten empfohlen. Diese Einteilung wurde ebenfalls in die derzeit vorliegende Fassung der BPMN-Spezifikation (OMG 2010a) übernommen. Sie werden dort als Prozess-Modellierungs-Übereinstimmungs-Klassen (Process Modeling Conformance Classes) bezeichnet.

Nach Silver 2009 (S. 7, S. 25ff) umfasst Stufe 1 eine möglichst einfache Darstellung des Prozessflusses unter Verwendung der grundlegendsten BPMN-Artefakte, die aus klassischen Flussdiagrammdarstellungen bekannt sind, um typische Abfolgen von Aktivitäten zu modellieren und diesen Aktivitäten Prozessrollen zuzuweisen. Allein dadurch ist eine kurze Einarbeitungszeit und schnelle Anwendbarkeit auch unter Nicht-BPMN-Spezialisten gewährleistet. Als Ergebnis ergeben sich einfache Modelle, die leicht zu verstehen und gut zur Kommunikation zwischen den Projektbeteiligten geeignet sind. Diese Stufe entspricht weitestgehend den Anforderungen, die eingangs als bedeutend für die Spezifikation einer Verbundanwendung eingestuft wurden und richtet sich primär an BPM-Berater und Prozessverantwortliche in den Fachabteilungen.

Stufe 2 nutzt die weitergehenden, leistungsstarken BPMN-Elemente, die eine sehr detaillierte Prozessmodellierung einschließlich der Ausnahme- und Ereignisbehandlung erlauben (Silver 2009, S.7, S. 59ff). Stufe 2 ist allerdings noch nach wie vor fachlich orientiert und die resultierenden Prozessmodelle sind noch immer nicht ausführbar (Silver 2009, S. 7). Sie können aber zur Analyse der Prozessperformanz aufgrund von Simulationen herangezogen werden. Außerdem stellen sie die fachliche Sicht auf einen in einem BPM-System (BPMS: Business Process Management System) ausführbaren Prozess dar. Es fehlen allerdings noch die technischen Details, die eine Ausführung ermöglichen. Stufe 2 wird in den meisten Fällen von Unternehmensarchitekten/-analysten bzw. dedizierten Prozessanalysten umgesetzt.

Stufe 3 schließlich ergänzt die bereits angesprochenen technischen Details im Prozessmodell der Stufe 2, die letztendlich zu einer Ausführbarkeit des Modells in einer

Prozess-Maschine führen (Silver 2009, S. 8, S. 189ff). Dies ist wohl eine der wichtigsten Unterscheidungen von BPMN zu anderen Modellierungsnotationen wie den ereignisgesteuerten Prozessketten (EPK) oder auch den UML-Aktivitätsdiagrammen: die Zugrundelegung einer kontrollierenden Prozess-Engine. Silver 2009 (S. 8) weist in diesem Zusammenhang darauf hin, dass die BPMN Teil eines größeren OMG-Ziels ist: der modellgetriebenen Architektur (MDA – model-driven architecture), in der ausführbare Systeme durch grafische Modelle gesteuert werden und nicht durch Code (Miller und Mukerji 2003). Stufe 3 richtet sich folglich eindeutig an Entwickler.

Kernpunkt des Silver'schen Vorgehensmodells ist die der hierarchischen Erweiterung (hierarchical expansion – Silver 2009, S. 14). Ausgehend von einem grobgranularen Ende-zu-Ende-Prozessmodell wird durch die in der BPMN unterstützten Unterprozesse eine weitere Verfeinerung erreicht, indem sie hierarchisch ineinander verschachtelt werden. Dadurch entstehen übersichtlichere Modelle, die je nach Bedarf vertieft betrachtet werden können und dabei gleichzeitig verschiedene Sichten für die verschiedenen Rollen auf das Modell ermöglichen: von der Sicht für den Prozessverantwortlichen auf der fachlichen Seite über die des Prozessanalysten bis hin zum Entwickler – ein Prozessmodell, das sämtlichen Anforderungen gerecht wird. Unterstützt durch entsprechende Werkzeuge, die eine Navigation entlang der Hierarchie ermöglichen, wird durch einen solchen Ansatz die Analyse von Prozessen, ihr Aufbau und ihre Umsetzung zusätzlich vereinfacht.

Ein weiterer Vorteil der hierarchischen Schachtelung von Unterprozessen liegt in der Wiederverwendbarkeit von Teilprozessen in anderen Szenarien. Ein Unterprozess kapselt demnach einen wohldefinierten, in sich abgeschlossenen betriebswirtschaftlichen Funktionsumfang, und er kann demnach auch als Komponente in einem SOA-Umfeld angesehen werden. BPMN forciert folglich die Komponentisierung von Prozessen/Teilprozessen und damit das Denken in wieder verwendbaren Einheiten. Aufgrund der vielfältigen Vorteile dieses Vorgehensmodells ist sie daher auch bereits während der Spezifikationsphase einzusetzen.

Da Stufe 1 am ehesten dem Detaillierungsgrad entspricht, der zur Spezifikation von Verbundanwendungen benötigt wird, lohnt sich hier ein genauerer Blick. Dabei ergeben sich interessante Übereinstimmungen mit den bereits aufgeführten Empfehlungen. Silver schlägt die folgenden 5 Schritte vor (Silver 2009, S. 35ff):

- Schritt 1: Definition des Umfangs des Prozesses, welche Funktionalität wird also durch den Prozess abgedeckt (Silver 2009, S. 35)?
- Schritt 2: Erstellung eines Top-Level-Modells für den „Happy Path“ (Silver 2009, S. 36)
- Schritt 3: Ergänzung des Top-Level-Modells um Pfade zur Behandlung von Ausnahmen (Silver 2009, S. 39)
- Schritt 4: Erweiterung des Modells um Unterprozesse zur Veranschaulichung weiterer Details auf Kind-Ebene (Silver 2009, S. 41)
- Schritt 5 (optional): Ergänzung um Nachrichtenflüsse zu externen Pools (Silver 2009, S. 43)

Dabei wird bei der Verwendung mehrerer Pools in Schritt 5 schon ein erhöhtes BPMN-Verständnis vorausgesetzt, das so sicherlich nicht verallgemeinert und vorausgesetzt werden kann. Freund, Rücker & Henninger (2010) lehnen daher die Verwendung mehrerer Pools aufgrund der erhöhten Komplexität für Fachabteilungen ab. Letztendlich müssen die Unternehmen individuell entscheiden, welchen Weg sie diesbezüglich gehen wollen. Eine Reduzierung der zu verwendenden Konstrukte ist aber auf alle Fälle durchzuführen, doch sollte dies der Entscheidung jedes einzelnen Unternehmens überlassen bleiben. Wie immer diese auch ausfällt, als Ergebnis liegen die Prozesse der Verbundanwendung in Form von BPMN-Modellen vor, auf deren Basis die nun folgenden Punkte behandelt werden können.

#### 3.2.2.4 Detailinformationen zu den Prozessschritten

Die im vorangegangenen Abschnitt erarbeiteten Prozessschritte müssen nun noch weiter verfeinert werden. Dabei geht es im Wesentlichen um die Frage, welche Funktionalität wie durch wen erbracht wird. Anhand der folgenden Checkliste, die für jeden Prozessschritt anzuwenden ist, lassen sich die für die Implementierung essenziellen Informationen identifizieren:

- Beschreibung des betriebswirtschaftlichen Hintergrunds des Schrittes: welchen Beitrag leistet er für den Gesamtprozess? Welche Geschäftslogik wird ausgeführt? Auf welchen Daten arbeitet er? Welche Aktionen werden auf den Daten ausgeführt (Lesen, Schreiben, Löschen)? Ist der Schritt auf Services angewiesen? Wenn ja, welche betriebswirtschaftliche Funktionalität wird von dem Service erwartet und welche Daten sind dafür aus Sicht der Verbundanwendung notwendig?
- Ist der Schritt interaktiv oder automatisiert? Im Falle eines interaktiven Schrittes ist zudem anzugeben, welche Eingaben erwartet werden.
- Ist der Schritt gegebenenfalls mehrfach auszuführen? Falls ja: ist die Ausführungsanzahl vorher bekannt oder wird sie erst durch die Erfüllung einer Bedingung beendet? Können bei bekannter Ausführungsanzahl die Schritte gleichzeitig (also parallel) ausgeführt werden oder ist auf eine strenge Sequenzialisierung zu achten?
- Können während der Aktivitätsausführung Ausnahmesituationen entstehen, auf die entsprechend reagiert werden muss? Was sind das für Ausnahmen?
- Bei interaktiven Schritten: welche Prozessrolle übernimmt diese Aufgabe? Gibt es Zeitbeschränkungen zu beachten? In diesem Zusammenhang ist es von Bedeutung festzulegen, bis zu welchem Zeitpunkt die Aktivität spätestens begonnen und bis zu welchem Zeitpunkt sie spätestens abgeschlossen sein muss. Werden diese Zeiten überschritten ist ebenfalls anzugeben, wie in solchen Fällen reagiert werden soll: gibt es einen separaten Eskalationspfad, dem gefolgt werden soll und wenn ja, soll dieser die auslösende Aktivität unterbrechen oder nicht?
- Welche Zustände werden ggf. aus dem Prozessschritt zurückgegeben? Je nach zurückgegebenen Zustand ist im weiteren Prozessfluss entsprechend zu reagieren. Dies muss sich im Prozessfluss widerspiegeln.

#### 3.2.2.5 Beschreibung des Datenflusses innerhalb des Prozesses (Prozesskontext)

Während der Ausführung von Prozessen werden unterschiedlichste Informationen und Daten behandelt und gesammelt. Von daher ist nicht nur die logische Abfolge der Prozessschritte bei der Betrachtung von Prozessen wichtig, auch der Nachrichtenfluss

zwischen den einzelnen Aufgaben muss bedacht werden. In der Spezifikation einer Verbundanwendung sind demnach Daten, die während der Prozessausführung den einzelnen Aktivitäten zur Verfügung stehen, explizit anzugeben. Dabei ist deren temporärer Charakter zu beachten: endet die Prozessinstanz, so sind diese Prozessdaten, auch Prozesskontext genannt, unwiederbringlich verloren. Sollen sie hingegen die Prozessinstanz überleben, so ist auch dies explizit in der Spezifikation anzugeben und deutlich von den Prozesskontext-Daten zu separieren. Die Darstellung der Daten in der Spezifikation kann beispielsweise in Form von Klassendiagrammen erfolgen. Dabei sind die für den Nachrichtenfluss benötigten Attribute zu Objekten zusammenzufassen. Auch die Beziehungen zu anderen Objekten sind darzustellen. Welche Details hierbei eine Rolle spielen, wird im Kapitel 3.2.4 (Geschäftsobjekte) weiter vertieft.

### *3.2.3 Ausnahmebehandlung*

Eine Berücksichtigung der während der Prozessdurchführung auftretenden Ausnahmen ist von fundamentaler Bedeutung. Es zwingt die Fachabteilungen, sich auch mit möglichen Fehlerszenarien auseinanderzusetzen. Viele der in der Praxis anzutreffenden Modelle befassen sich nahezu ausschließlich mit dem „Happy Path“, obwohl gerade die fehlerhaften Fälle in der Regel bei der Ausarbeitung von Prozessmodellen den größten Aufwand erfordern (Allweyer 2009a). Dabei geht es in diesem Teil der Spezifikation konkret um die Behandlung rein betriebswirtschaftlicher Ausnahmefälle. So ist anzugeben, unter welchen Umständen an welcher Aktivität Ausnahmesituationen auftreten können und wie zu reagieren ist. Bei automatisierten Schritten (Service-Aufgabe in BPMN) ist zudem zu ergänzen, ob ggf. eine Wiederholung des Aufrufs durchgeführt wird und wenn ja, wie häufig und in welchen Abständen dies erfolgen soll. Auch ein kompletter Prozessabbruch ist denkbar, wobei allerdings auch in diesem Fall nach einer Lösung gesucht werden muss, wie weiter zu verfahren ist und wer darüber zu informieren ist.

Folgt man dem hierarchischen Ansatz, so wie er im vorangegangenen Abschnitt vorgeschlagen wurde, so ist auch über die Kommunikation von Ausnahmen aus einem Unterprozess heraus an die übergeordnete Prozessinstanz nachzudenken. Treten Ausnahmen in Unterprozessen auf, so ist dabei ein besonderes Augenmerk auf die Art und Weise zu legen, wie dieser Zustand an den übergeordneten Prozess weitergegeben wird, damit dieser wiederum geeignet reagieren kann. Dabei stehen dem Unterprozess drei Möglichkeiten zur Verfügung:

- Anzeige des Fehlers durch Setzen eines Fehlerstatus im Unterprozess. In BPMN stehen dazu End-Ereignisse zur Verfügung, deren Zustand der übergeordnete Prozess (Vaterprozess) durch ein nachgelagertes Gateway auswerten kann.
- Werfen einer Ausnahme. Da es sich um betriebswirtschaftliche Ausnahmen handelt, empfiehlt sich in BPMN das Eskalations-Ereignis. Der übergeordnete Prozess fängt dieses Ereignis durch ein an den Unterprozess geheftetes Eskalations-Ereignis ab und folgt dem daran modellierten Sequenzfluss.
- Setzen einer Variablen im Prozesskontext des übergeordneten Prozesses. Dies ist allerdings nur möglich, sofern der Unterprozess in dem übergeordneten Prozess eingebettet ist und nicht als eigenständiger Prozess aufgerufen wird.

Von diesen Alternativen ist die zweite Möglichkeit eindeutig zu bevorzugen. Es können dadurch unterschiedlichste Fehlerfälle angezeigt und entsprechend dediziert darauf reagiert werden. Die Modelle bleiben kompakt und trennen sauber den Sequenzfluss für den Normalfall von den Sequenzflüssen der Ausnahmebehandlungen.

### *3.2.4 Geschäftsobjekte*

Verbundanwendungen arbeiten auf einem bestimmten Satz von Daten, die sich zu Geschäftsobjekten wie beispielsweise Kunde, Auftrag oder Lieferant zusammenfassen lassen. Geschäftsobjekte werden in der Regel nicht isoliert innerhalb der Verbundapplikation eingesetzt, sondern stehen vielmehr in Relation zu anderen Objekten. Folglich ist es Gegenstand der Spezifikation von Geschäftsobjekten, sowohl die Attribute der einzelnen Objekte als auch die Verbindungen untereinander anzugeben. Auch hier wird vollständig top-down vorgegangen: es werden lediglich Objekte und Attribute aufgeführt, die aus Sicht der Composite eine Rolle spielen. Datenstrukturen aus existierenden Anwendungen werden nicht wiederverwendet, da dadurch das Unabhängigkeitsgebot verletzt würde. Zur Detailbeschreibung eines Geschäftsobjekts gehören beispielsweise:

- sprechender Attributname (z.B. Vorname, Postleitzahl, Gewicht)
- Datentyp (z.B. String, Integer, Datum, Boolean,...)
- Pflichtfeld (ja/nein)

- Längenbeschränkungen (z.B. 35 Zeichen)
- Wertebereich (z.B. von 0 bis 999.999)
- Eingabemasken (z.B. Produktnummern, die einem bestimmten Muster folgen, wobei für die Masken vorgegebene Symbole die möglichen Eingaben charakterisieren)
- Speicherort der Felder: welche Erwartungen hat die Composite hinsichtlich der Persistenz der einzelnen Attribute – lokal (innerhalb der Verbundanwendung) oder remote (d.h. in einem existierenden System der IT-Landschaft)?
- Abhängigkeiten zu bzw. Zusammenhänge mit anderen Feldern (z.B. wenn Feld xyz den Wert abc enthält, so muss das Feld uvw in einem bestimmten Wertebereich liegen)

Nach der Spezifikation der Geschäftsobjekte mit ihren Attributen müssen noch die Relationen angegeben werden. Es genügt, Beziehungstyp (Assoziation und Komposition) sowie Kardinalität festzulegen. Der Beziehungstyp zwischen zwei Objekten hängt davon ab, ob das Kind-Objekt beim Löschen seines Vaters ebenfalls gelöscht werden soll. Ist dem so, so handelt es sich um eine Komposition (z.B. *Auftragsposition* als Kind-Objekt zu seinem Vaterobjekt *Auftrag*), andernfalls um eine Assoziation (z.B. *Kunde* als Kind-Objekt zu *Auftrag* als Vaterobjekt). Bei den Kardinalitäten sind folgende Unterscheidungen üblich:

- 0..1: Es *kann* eine Beziehung zu *genau einem* Objekt bestehen.
- 1..1: Es *muss* eine Beziehung zu *genau einem* Objekt bestehen.
- 0..n: Es *kann* eine Beziehung zu *mehreren* Objekten bestehen
- 1..n: Es *muss mindestens eine* Beziehung zu einem Objekt bestehen

m..n-Beziehungen werden entsprechend durch zwei 0..n- bzw. 1..n-Beziehungen in den betroffenen Objekten definiert. Zum besseren Verständnis der Geschäftsobjekte ist die Angabe einiger informativer Daten anzuraten. Hierzu zählen beispielsweise Informationen über die zu erwartende Anzahl der Objekte, die Zugriffshäufigkeit zur Laufzeit als auch die primäre Nutzungsart (lesend, schreiben, suchend). Derartige De-

tails helfen bei der späteren Umsetzung bei der Wahl der richtigen Technologie.

### 3.2.5 Benutzeroberflächen

Benutzeroberflächen sind für Verbundanwendungen aufgrund ihrer kollaborativen Ausrichtung von besonderer Bedeutung. Bereits in dieser sehr frühen Phase müssen die Eingabemasken an den Bedürfnissen der Endanwender optimal ausgerichtet werden. In Zusammenarbeit mit den Benutzern werden Oberflächen-Prototypen entwickelt. Dieses Vorgehen erhöht später die Akzeptanz bei der Produktivsetzung. Außerdem reduziert sie Kosten, da Probleme rechtzeitig erkannt und in frühen Phasen mit überschaubaren Aufwand behoben werden können. Dieses als Usability Engineering bezeichnete Verfahren und die damit einhergehenden Vorteile werden beispielsweise in Eller 2009 detailliert diskutiert.

Ziel ist die bestmögliche Unterstützung des Endanwenders in seiner Prozessrolle. Sämtliche zur Erledigung seiner Aufgabe benötigten Daten müssen anwendergerecht aufbereitet und präsentiert werden. Dabei sind die Masken gleichzeitig für effizientes Arbeiten auszurichten: mit einem Minimum an Schreib- und Klickaufwand müssen die erwarteten Informationen eingebbar sein. In ständigen Review-Zyklen mit dem Benutzer sind die Vorschläge zu verfeinern. Eine derart erarbeitete Oberfläche kann nun als Screenshot in die Spezifikation übernommen werden. Anschließend ist jedes Maskenfeld mit Daten zu assoziieren. Hierbei wird Bezug auf die bereits definierten Geschäftsobjekte genommen, wodurch sich automatisch wichtige Informationen wie Datentyp, Pflichtfeld und Eingabebeschränkungen (Wertebereiche, Feldmasken) ergeben. Darüberhinausgehende Eigenschaften, wie z.B. die visuelle Repräsentation der Daten über Charts, Säulen, Tortendiagrammen oder Kurven sind genauso explizit zu ergänzen wie die Zuweisung von User-Interface-Komponenten wie Radio-Button, Checkboxen, Dropdown-Listen, Kalender oder Bäumen zu einzelnen Feldern. Schließlich darf die Änderbarkeit der Felder nicht fehlen: handelt es sich um ein Read-Only-Feld oder sind Eingaben erlaubt?

Die Bearbeitungszeit von Masken kann reduziert werden, indem die Eingabe von Feldern durch Wertehilfen unterstützt wird. Insbesondere bei komplexen Zusammenhängen zwischen Feldern erleichtert diese Maßnahme die Maskenbearbeitung ungemein. Sind Wertehilfen für bestimmte Felder/Feldergruppen vorgesehen, so muss geklärt werden, wie die erlaubten Werte konkret zu bestimmen sind. In der Regel wer-

den mögliche Eingaben erst zur Laufzeit durch Service-Aufrufe ermittelt, auf die in der Spezifikation explizit verwiesen wird und die ebenfalls mit Beschreibung der von ihnen abgedeckten Funktionalitäten aufgeführt sein müssen. Wie dies im Detail zu erfolgen hat, wird im nächsten Abschnitt diskutiert.

Ähnlich wie bei der Ermittlung erlaubter Werte wird auch bei Feldprüfungen für gewöhnlich auf Services zurückgegriffen. Alternativ sind allerdings auch regelbasierte Ansätze denkbar. In diesem Fall sind die Regeln zur Feldüberprüfung explizit anzugeben. Andernfalls genügt erneut ein Verweis auf den dafür vorgesehenen Dienst.

Grundsätzlich ist die Auflistung möglicher erlaubter Eingaben für bestimmte Felder immer sinnvoll. Sie erleichtern dem Architekten die Umsetzung der Anforderungen und beugen Rückfragen vor.

### *3.2.6 Dienste*

Services sind das Fundament einer jeden Verbundanwendung. Im Rahmen dieses Kapitels zur Spezifikation von Composites wurde bereits des Öfteren auf Services hingewiesen: so wurde beispielsweise bei der Ausarbeitung der Detailinformationen zu den Prozessschritten als auch bei der Definition der Benutzeroberflächen ihre Verwendung berücksichtigt. Sie übernehmen innerhalb einer Composite gleich zwei Funktionen.

- Bereitstellung neuer betriebswirtschaftlicher Logik, die speziell für die Composite aufgrund fehlender Alternativen neu entwickelt werden muss und als Service zur Verfügung steht.
- Kapselung und damit Abstraktion existierender betriebswirtschaftlicher Funktionalitäten

Unabhängig vom Einsatz des Dienstes ist innerhalb der Spezifikation das Aussehen der Schnittstelle entscheidend. Dabei ist die Schnittstelle ausschließlich aus Sicht der Verbundanwendung zu definieren und ist daher rein fachlich motiviert. Eine Berücksichtigung existierender Schnittstellen von Services aus Anwendungssystemen wie ERP- oder SRM-Lösungen wird bewusst nicht verfolgt, da sich daraus Abhängigkeiten ergeben, die in Verbundanwendungen zu vermeiden sind. Dieser Ansatz unterscheidet sich daher grundsätzlich von Vorgehensweisen, bei denen zu einer direkten

Verwendung von Backend-Services geraten wird. Folglich sind Dienste für deren Verwendung in Composites wie folgt zu spezifizieren:

- Vergabe eines sprechenden Servicenamens durch Verwendung bestimmter Namenskonventionen, die die betriebswirtschaftliche Funktion des Services an seinen Namen erkennen lassen. Üblicherweise werden Verben mit Nomen verbunden, wie beispielsweise bei dem Service *createProject* oder *calculateFunding*.
- Auflistung sämtlicher Ein- und Ausgabeparameter des Dienstes einschließlich der zu verwendenden Datentypen. Idealerweise kann auf Attribute der bereits spezifizierten Geschäftsobjekte verwiesen werden, so dass neben dem eigentlichen Datentyp auch von den dort definierten Einschränkungen und Wertebereichen profitiert werden kann.
- Kurzbeschreibung der zu erbringenden betriebswirtschaftlichen Funktionalität bzw. Geschäftslogik. Hiermit wird die Semantik des Service festgelegt.
- Vermerk, ob die abzudeckende betriebswirtschaftliche Funktionalität durch eine externe Implementierung erbracht wird oder ob es sich um einen composite-spezifischen Dienst handelt, der neu zu entwickeln ist. Handelt es sich bei der Schnittstelle um eine Service, der extern zu erbringen ist, so wird durch die Interface-Definition gleichzeitig der Service-Kontrakt aus Abbildung 2 zwischen Composite und der externen Welt festgelegt.

- Wird der Service neu entwickelt, so ist bei verändernden Diensten wie Aktualisierungen und Löschungen anzugeben, wie sich der Dienst bei konkurrierenden Zugriffen zu verhalten hat: soll bei Schreibkonflikten der jeweils letzte Aufruf alle vorangegangenen Änderungen überschreiben können oder soll, wie bei optimistischen Sperrvorgängen üblich, das Erkennen eines Konflikts durch eine Ausnahme angezeigt werden? Auch hier können Namenskonventionen behilflich sein. So unterscheidet die SAP bei der Definition ihrer Enterprise Services zwischen der Verwendung der Präfixe *change* und *update*: der Dienst ***change****Customer* würde eine Änderung ohne Konflikterkennung durchführen, während ***update****Customer* einen solchen Konflikt erkennen und durch eine Ausnahme anzeigen könnte. Eine entsprechende Reaktion des Aufrufers bei Konflikterkennung ist in dem letztgenannten Fall möglich und muss ebenfalls spezifiziert werden.
- Neben den Ausnahmen, die bei Konflikten geworfen werden, zeigen Services in der Regel auch andere außergewöhnliche Situationen während ihres Ablaufs durch Ausnahmen an. Da Ausnahmen bereits spezifiziert wurden, kann daher auf sie in dem dazugehörigen Abschnitt der Spezifikation verwiesen werden. Durch derartige Querverweise können zudem Lücken in der Auflistung von Ausnahmen entdeckt und entsprechend korrigiert werden.
- Insbesondere bei der Spezifikation extern zu erbringender Dienste sind Angaben über nicht-funktionale Anforderungen, den sogenannten *Quality-of-Service*-Attributen (QoS), nicht zu vergessen. Diese Umfassen u.a. Informationen über die zu erwartende Belastung in Form von Aufrufmengen (z.B. Anzahl Aufrufe pro Zeiteinheit), Sicherheitsaspekte, die Verfügbarkeit und das Zeitverhalten. Der letztgenannte Aspekt ist hierbei von besonderer Bedeutung. Hierzu gehören Angaben, wie lange die Operation maximal dauern darf und wie zu reagieren ist, falls die vorgegebene Zeit überschritten wird: ist die Zeitüberschreitung kritisch, müssen also sämtlich laufende Aktivitäten unterbrochen und ein Fehler in Form einer Ausnahme an den Aufrufer zurückgemeldet werden, oder sollen die bereits angestoßenen Aufrufe weiterlaufen und parallel dazu lediglich ein technischer Administrator bzw. Fachexperte über eine mögliche Verzögerung informiert werden?

Eine derart erstellte Spezifikation bildet das Fundament für die weitere Implementierung der Verbundanwendung. Zusammenfassend ist der konsequente Top-Down-Ansatz wohl die markanteste Eigenschaft dieses Vorgehens: allein die betriebswirtschaftlichen Anforderungen formen die Prozesse, die detaillierten Prozessabläufe und daraus wiederum resultierend sowohl die Benutzeroberflächen, die Daten, auf denen operiert wird, als auch die verwendeten Services. Dabei spielen auch die verwendeten Datentypen eine wichtige Rolle. Soweit wurde lediglich darauf hingewiesen, dass Datentypen aus existierenden Backend-Systemen nicht wiederzuverwenden sind, da dadurch Abhängigkeiten zu genau diesen Systemen entstehen. Doch welche Alternative bietet sich dann an? Dies soll im nun folgenden Abschnitt diskutiert werden.

### *3.2.7 Bedeutung des kanonischen Datenmodells*

Ein Kernaspekt der Verbundanwendungsentwicklung beruht auf der Vermeidung von Abhängigkeiten, wie sie bei der unmittelbaren Verwendung externer Dienste auftreten. Werden genau diese Applikationen, aus denen die Schnittstellen ursprünglich importiert wurden, zu einem späteren Zeitpunkt einmal ersetzt, so zieht sich der Änderungsaufwand zur Anpassung an die geänderte Systemlandschaft durch die gesamte Verbundanwendung hindurch: von der Service-Seite über die Benutzeroberflächen bis in den Prozess hinein. Die Flexibilität der Composite wird eingeschränkt, die kurzfristige Anpassbarkeit an geänderte Rahmenbedingungen behindert. Damit einher geht aber noch ein weiterer gravierender Nachteil: werden mehrere Datentypen aus unterschiedlichsten Anwendungen in der Composite wiederverwendet, so sind Unverträglichkeiten zwischen den Datentypen unvermeidbar. Klassische Beispiele sind:

- Ein Kundenobjekt aus einer Anwendung A verwendet getrennte Felder für Vor- und Nachname, während das Kundenobjekt aus Anwendung B diese in einem Feld zusammenfasst.
- Es ist zwischen unterschiedlichen Datentypen zu vermitteln, so z.B. zwischen String und Integer. Sind dann noch unterschiedlichen Wertebereichen zu berücksichtigen, gestaltet sich die Lösung allein dieses Problems als recht aufwändig.
- Es muss eine Wertemapping vorgenommen werden: so liefert eine Anwendung den Inhalt 1 zur Kennzeichnung der Anrede „Frau“ und 2 für die Anrede „Herr“, während eine andere Anwendung genau die Klartexte verwendet.

Der Composite-Entwickler sieht sich zwangsläufig mit Datentyp-Transformationen konfrontiert, die es wiederum in einer Composite zu vermeiden gilt. Die Lösung liegt in der Verwendung eines kanonischen Datenmodells, das bereits bei der Spezifikation der Verbundanwendung berücksichtigt wird. Ein kanonisches Datenmodell ist ein universelles systemübergreifendes Datenmodell, das von den zu integrierenden Anwendungen vollständig unabhängig ist. Die Verwendung derartiger Modelle ist nicht neu (siehe Hohpe & Woolf 2004 oder Maier et. al. 2009a), allerdings erstreckt sich deren Verwendung nun neben einem Einsatz in Integrationsszenarien auch in die Verbundanwendung selbst hinein. Erreicht wird dadurch letztendlich eine lose Kopplung auf Datentypebene mit dem Ergebnis, dass innerhalb einer Composite Application auf einem harmonischen Datentypsystem gearbeitet wird und dadurch jegliche Arten von Transformationen vermieden werden. In den Ausführungen zur Spezifikation von Verbundanwendungen ist die Verwendung von Datentypen an mehreren Stellen aufgetreten:

- Bei der Definition des Prozesskontextes
- Bei der Festlegung der Daten für Benutzer-Interaktionen
- Bei der Beschreibung der Service-Schnittstellen
- Bei der Definition der Geschäftsobjekte

In all diesen Fällen ist das kanonische Datenmodell zu verwenden, das sich auf diese Weise über die gesamte Composite Application erstreckt. Hervorgehoben werden muss an dieser Stelle die besondere Bedeutung des kanonischen Datenmodells bei der Beschreibung der Service-Schnittstellen, deren betriebswirtschaftliche Logik von externen Systemen zu erbringen ist und damit die Serviceverträge aus Abbildung 2 repräsentieren. Diese Serviceverträge können anschließend auf vielfältige Weise in der Servicevertrag-Implementierungsschicht umgesetzt werden, so u.a. auch durch Vermittlung innerhalb eines Enterprise Service Bus (ESB). Wird nun auf dieser Ebene dasselbe kanonische Datenmodell verwendet, reduziert sich der Transformationsaufwand zusätzlich. Es ist heute in größeren Unternehmen üblich, ESB-Lösungen einzusetzen, so dass bereits ein kanonisches Datenmodell vorliegt. Da es sich bei Verbundanwendungen um vollständig neue Applikationen handelt, empfiehlt sich die Wie-

derverwendung des auf ESB-Ebene bereits definierten Datenmodells. Hohpe & Woolf 2004 weisen zu Recht auf die Vorteile des kanonischen Datenmodells auf ESB-Ebene hin, wenn man allein die Anzahl der zu erstellenden Mappings zwischen den unterschiedlichen Datenformaten berücksichtigt. Insbesondere bei einer steigenden Anzahl zu verknüpfender Anwendungen zahlt sich dieses Vorgehen aus. Bei bidirektionalen Punkt-zu-Punkt-Verbindungen benötigt man bei n Systemen, die mit m Systemen zu verbinden sind, n * m * 2 Mapping-Definitionen. Entsprechend sind bei n Systemen, die untereinander konnektiert sind, n * (n - 1) Mappings zu definieren. Die Kurve steigt also quadratisch. Durch Einführung eines kanonischen Datenmodells ändern sich die Zahlen wie folgt: bei der Kommunikation von n Systeme mit m anderen Systemen ergeben sich (n + m) * 2 Mappings und bei n untereinander verbundenen Systemen n * 2 Mappings. Es ergibt sich also ein linearer Verlauf. Darüber hinaus lassen sich folgende Vorteile zusammenfassen (aus Maier et. al. 2009a):

- Services, die auf dem kanonischen Datenmodell basieren, müssen die Datentyp-Dialekte der jeweiligen Kommunikationspartner nicht kennen. Genau dieses Ziel wird bei Verbundanwendungen verfolgt, da deren konkrete Partner sich erst bei einer Installation in einer Systemlandschaft ergeben. Der Service-Konsument, also die Composite, ist ergo vollständig vom internen Datendialekt des Service-Providers entkoppelt.
- Veränderungen am internen Datenmodell der Service-Implementierung beeinflussen den Service-Konsumenten nicht.
- Der Transformations-Overhead nimmt mit zunehmender Systemanzahl ab.
- Klare Trennung von Verantwortlichkeiten: die Datentypen sind eindeutig den Systemen und damit den Domänen und den dafür verantwortlichen Rollen zugeordnet.
- Ein kanonisches Datenmodell trägt zur Reduzierung von Missverständnissen bei, da sowohl die fachliche als auch die technische Seite über ein und dasselbe Domänenobjekt sprechen.

Dem gegenüber stehen allerdings auch einige Nachteile: so ist insbesondere die Erstellung eines kanonischen Datenmodells ein immer wiederkehrendes Problem. Hier empfiehlt sich die Orientierung an Standards. So haben sich in der SAP-Welt die so-

genannten Global Data Types (GDT) etabliert, die wiederum auf dem CCTS-Standard basieren (CCTS: Core Component Technical Specification, ein UN/CEFACT Standard – siehe auch UN/CEFACT 2003). GDTs sind SAP-weit einheitlich definiert und repräsentieren fachlich motivierte Datenmodelle. Sämtliche seitens der SAP veröffentlichten Enterprise Services (wohldefinierte, in sich abgeschlossene betriebswirtschaftliche Funktionalitäten, die als Web Services zur Verfügung gestellt werden) beruhen auf Global Data Types. Verwendet ein SAP-Kunde also derartige Enterprise Services und legt er sich zudem sowohl für seinen ESB als auch für die Verbundanwendung auf diese Datentypen fest, ist Mappingaufwand nur noch für Verbindungen mit Nicht-SAP-Systemen zu treiben – ein nicht zu unterschätzendes Einsparpotenzial für den Gesamtaufwand, der für die Erstellung von Composites zu erbringen ist. Details über die SAP-Methodologie zur Erstellung von Datenmodellen sowie die Dokumentation der GDTs finden sich in SAP 2009a bzw. SAP 2009b respektive.

Auch wenn das Modell nicht für jedes Unternehmen die bestmögliche Lösung darstellt, so kann es dennoch gut als Vorlage für die Definition eines eigenen Modells dienen. Alternativ kann auch auf branchenspezifische Datenmodellen zurückgegriffen werden. S.W.I.F.T. (Society for Worldwide Interbank Financial Telecommunication) für die Bankenbranche ist hier nur ein Beispiel. Die Open Applications Group (OAGi) hat mit OAGIS (Open Applications Group Integration Specification – siehe auch OAGIS 2009) einen branchenübergreifenden Standard hervorgebracht, der ebenfalls sehr gut als Ausgangspunkt für ein eigenes Datenmodell verwendet werden kann.

Rein technisch gesehen bedeutet die Verwendung eines anderen Datenmodells in der Composite im Vergleich zu dem im ESB verwendeten Datenmodells ein zusätzlicher Transformationsaufwand während des Nachrichtentransfers: obwohl die Anzahl der Mapping-Definitionen selbst in Summe reduziert wird (s.o.), nimmt die Anzahl der auszuführenden Mappings während der Laufzeit zu. Schließlich muss jede Nachricht, die eine Verbundanwendung verlässt, in das Datenformat des ESB's umgesetzt und anschließend wiederum in das Datenformat der Zielanwendung transformiert werden. Der Rückweg gestaltet sich entsprechend aufwändig. Unter diesen Umständen ist auch die Empfehlung zu verstehen, von vornherein auch in der Composite Application sich des Datenformats des EBS's anzunehmen. Jede durchzuführende Transformation benötigt also Zeit und führt folglich zu Verzögerungen. Für Hochlastszena-

rien ist dieses Vorgehen daher nicht zu empfehlen. In diesen Fällen bleibt als einzige Alternative die direkte Transformation von dem Ausgangsformat in das Zielformat. Derartige Konflikte/Spannungen (und damit verbunden die zu treffenden Entscheidungen) zwischen Komfort, besserer Wartbarkeit sowie gesteigerter Flexibilität auf der einen und reduzierter Performanz sowie gesteigerter Komplexität auf der anderen Seite, sind für Verbundanwendungen aufgrund ihrer ganz spezifischen Rolle als Lösungen, die in einem höchst heterogenen Umfeld agieren, ganz normal. Grundsätzlich überwiegen allerdings die Vorteile, so dass bei Entscheidungen stets die bessere Wartbarkeit und die gesteigerte Flexibilität im Vordergrund stehen sollten. Einzige Ausnahme davon sind Performanz-Aspekte. Sind die Anforderungen an die Performanz mit der angestrebten Architektur nicht zu erreichen, muss auf Alternativen umgestellt werden, auch wenn dies mit einem Bruch der Architekturgrundsätze verbunden ist. Aber Anwendungen mit unverhältnismäßig hohen Antwortzeiten werden von Kunden einfach nicht akzeptiert. Bevor allerdings gleich die gesamte Architektur in Frage gestellt wird, sollte zumindest nach Alternativen gesucht werden, wie mit der existierenden Architektur doch noch Verbesserungen zu erzielen sind. So ist insbesondere nach Parallelisierungsmöglichkeiten (horizontale Skalierung) zu suchen, die sich insbesondere für zustandslose Aktivitäten wie die oben erwähnten Mappings anbieten. Moderne Multicore-Maschinen können derartige Dienste ausgesprochen gut parallelisieren mit entsprechenden Performanz-Verbesserungen als Folge.

## 3.3 Einführung in die Grundarchitektur von Verbundanwendungen

Nachdem im vorangegangenen Abschnitt die wesentlichen Anforderungen an eine Verbundanwendung zusammengetragen wurden, wird im Folgenden die Grundarchitektur weiter verfeinert. Zur Erinnerung wird daher nochmals auf die schematische Darstellung der Verbundanwendung (Abbildung 9) zurückgegriffen:

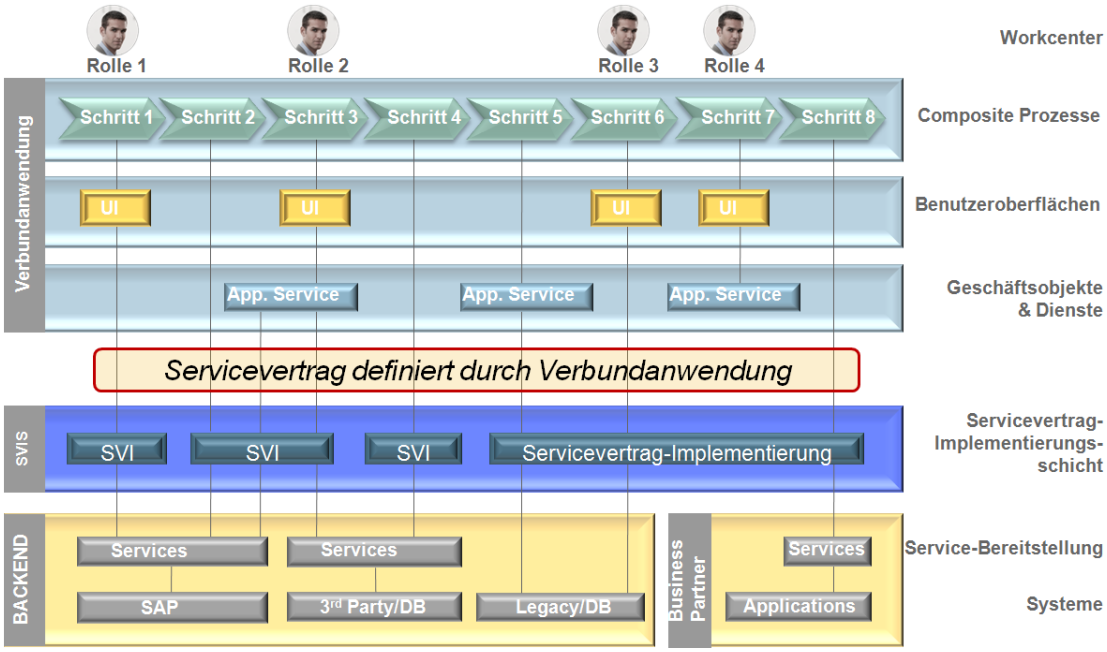



***Abbildung 9: Schematische Darstellung einer Verbundanwendung (Grimm, Speck, Stiehl 2010)***

Besonders augenfällig und damit charakteristisch für eine Verbundanwendung ist die vollständige Trennung zwischen der eigentlichen Anwendung (im oberen Teil der Abbildung mit „Verbundanwendung“ bezeichnet), dem Servicevertrag und dessen Implementierung in der Servicevertrag-Implementierungsschicht. Durch die in Abschnitt 3.2 gesammelten Informationen wird ausschließlich die Verbundanwendung selbst adressiert. Prozesse, Benutzeroberflächen sowie Geschäftsobjekte und Dienste liegen zum gegenwärtigen Zeitpunkt fest. Im Weiteren stehen daher das Zusammenwirken von Verbundanwendung, dem nach außen gerichteten Servicevertrag sowie dessen konkrete Umsetzung im Mittelpunkt.

In Abbildung 9 verlaufen offensichtlich sämtliche Kommunikationswege zwischen Verbundanwendung und den in der Systemlandschaft betriebenen Backend-Systemen über die Servicevertrag-Implementierungsschicht. Dabei deckt die Verbundanwendung selbst ausschließlich die betriebswirtschaftlichen Funktionalitäten ab, die einem Unternehmen Wettbewerbsvorteile gegenüber der Konkurrenz bringen, während die Servicevertrag-Implementierungsschicht sich neben der Implementierung der von der Composite extern benötigten Dienste auch der technischen Details annimmt: physikalische Verbindungen zu den angeschlossenen Backend-Systemen, Unterstützung verschiedenster Protokolle, Vermittlung zwischen den unterschiedlichen Datentypen, Fehlerbehandlung auf technischer Ebene usw.

Dieser Ansatz der vollständigen, kompromisslosen Trennung ist von fundamentaler Bedeutung für Composite Applications und liegt in der kompletten Entkopplung von Composite und Backend begründet: niemals werden in einer Verbundanwendung Aufrufe direkt an konkrete Systeme abgesetzt. Kommunikationspartner ist stets die Servicevertrag-Implementierungsschicht über den aus Sicht der Verbundanwendung definierten Servicevertrag, der somit eine Abstraktion gegenüber den real existierenden Systemen darstellt. Auf diese Weise können keinerlei Abhängigkeiten zwischen Verbundanwendung und Backend-Systemen entstehen. Die Verbundanwendung selbst wird dadurch „zeitloser“ (Aigner 2010). Als Ergebnis wird eine Unabhängigkeit der unterschiedlichen Lebenszyklen dieser beiden Welten erreicht: die durch Verbundanwendungen adressierten innovativen Lösungen können in einem wesentlich höheren Tempo und damit in deutlich kürzeren Zyklen vorangetrieben werden. Die Kernanwendungen in den Altsystemen bleiben hingegen über einen erheblich längeren Zeitraum hinweg stabil. Damit wird ein weiteres wichtiges Ziel für die Betreiber der IT-Landschaften innerhalb der Unternehmen erreicht: keine zu häufigen Änderungen an den Kernprozessen, die die Hauptlast einer geordneten Geschäftsabwicklung zu tragen haben und über die Tag für Tag Millionen von Transaktionen innerhalb kürzester Zeit abzuwickeln sind. Dies wird zudem durch die Nicht-Invasivität der Verbundanwendung unterstützt: seitens der Backend-Systeme bereitgestellte Dienste werden über die Servicevertrag-Implementierungsschicht so verwendet, wie sie originär vom Provider vorgesehen wurden. Es werden keinerlei Änderungen an den Backend-Systemen vorgenommen. Ergibt sich ein Delta zwischen den Erwartungen seitens der Composite und der durch die Backends bereitgestellten Funktionalität, so muss diese in der Servicevertrag-Implementierungsschicht umgesetzt werden.

Mit anderen Worten: es wird eine lose Kopplung zwischen der Verbundanwendung und den servicebereitstellenden Backend-Systemen angestrebt. Dabei stellt die Datentyp-Abhängigkeit beispielsweise eine Dimension innerhalb des Spektrums lose gekoppelter Systeme dar. Weitere Aspekte werden in dem nun folgenden Abschnitt diskutiert.

### *3.3.1 Lose Kopplung*

Lose Kopplung stellt wohl eine der wichtigsten Aspekte nicht nur bei der Konstruktion von Verbundanwendungen sondern allgemein bei dem Entwurf großer ver-

teilter heterogener Systeme dar. Letztendlich steht dabei generell die Reduzierung von Abhängigkeiten im Mittelpunkt. Allerdings gilt es dabei, mehrere Dimensionen der Abhängigkeit zu unterscheiden. Behilflich dabei ist die folgende aus Josuttis 2008 entnommene Tabelle möglicher Formen der losen Kopplung, die ursprünglich in Krafzig et.al. 2004 eingeführt und diskutiert wurde.

| | **Enge Kopplung** | **Lose Kopplung** |
|---|---|---|
| Physische Verbindung | Punkt-zu-Punkt | Über Vermittler |
| Kommunikationsstil | Synchron | Asynchron |
| Datenmodell | Komplexe gemeinsame Typen | Nur einfache gemeinsame Typen |
| Typsystem | Streng | Schwach |
| Binding | Statisch | Dynamisch |
| Plattformspezifika | Stark bzw. viel | Schwach bzw. wenig |
| Interaktionsmuster | Über komplexe Objektbäume navigieren | Datenzentrierte autonome Nachrichten |
| Transaktionssicherheit | 2PC (Two-Phase-Commit) | Kompensation |
| Kontrolle fachliche Logik | Zentrale Kontrolle | Verteilte Kontrolle |
| Versionierung | Explizite Upgrades | Implizite Upgrades |

***Tabelle 5: Formen loser Kopplung (aus Josuttis 2008, S. 48)***

In den folgenden Ausführungen wird der Schwerpunkt der Diskussion von Tabelle 5 auf die Architekturauswirkungen für Verbundanwendungen gelegt. Weitere Details zur Bedeutung im Rahmen von SOA sei auf Josuttis 2008 und Krafzig et.al. 2004 verwiesen.

#### 3.3.1.1 Physische Verbindung

Dieser Punkt adressiert die Frage, ob Verbundanwendungen direkt mit ihren Kommunikationspartnern physisch verbunden sein sollen, ob ihnen also deren konkrete physische Netzadressen bekannt sein sollen oder ob die Verbindung über einen Vermittler zu erfolgen hat (Josuttis 2008, S. 55/56). Aufgrund der Definition von Verbundanwendungen verbietet sich zwangsläufig eine direkt physische Verbindung, da sich dadurch unmittelbar Abhängigkeiten zu technischen Details wie beispielsweise der konkreten IP-Adresse oder dem zu verwendenden Protokoll ergeben. Derartige Infor-

mationen sind gemäß der Architekturempfehlung aus Abbildung 9 in die Servicevertrag-Implementierungsschicht zu verlagern. Es findet also, wie in der Tabelle für die lose Kopplung vorgeschlagen, eine Vermittlung statt. Diese muss allerdings nicht zwangsläufig über eine Message-Oriented Middleware (MOM) oder einen ESB erfolgen, so dass mit der Einführung einer Verbundanwendung nicht gleichzeitig die Installation eines entsprechenden Messaging-Produkts verbunden ist. Es können auch leichtgewichtige Lösungen wie beispielsweise Java SE, Java EE, C# oder jede anderen Programmiersprache verwendet werden, solange der Servicevertrag gemäß der Vorgabe der Composite erfüllt wird.

#### 3.3.1.2 Kommunikationsstil

Bei der Diskussion des Kommunikationsstils ist zu beachten, dass in diesem Abschnitt ausschließlich die externe Kommunikation der Verbundanwendung mit ihrer Servicevertrag-Implementierungsschicht gemeint ist. Dabei wird zwischen synchroner und asynchroner Kommunikation unterschieden. Synchron bedeutet in diesem Zusammenhang das Warten des Senders auf eine Antwort des Empfängers, um diese anschließend sofort weiterzuverarbeiten. In der Literatur wird diese Art der Kommunikation auch als Request-Reply (Hohpe & Woolf 2004) bzw. Request-Response (Josuttis 2008, S. 154) Message Exchange Pattern (MEP) bezeichnet. Wichtig dabei ist, dass in einem Aufruf des Senders sowohl die ausgehende als auch die eingehende Nachricht abgewickelt werden. Eine synchrone Schnittstellendefinition in WSDL (Web Services Description Language) umfasst daher sowohl Eingabe- als auch Ausgabeparameter.

Im Gegensatz dazu wird bei der asynchronen Kommunikation vom Sender eine Nachricht abgesetzt ohne explizit auf eine Bestätigung des Empfängers zu warten. Dementsprechend enthält die dazugehörige Schnittstellendefinition, wiederum in WSDL ausgedrückt, lediglich Eingabeparameter. Diese Unterscheidung ist von daher wichtig, als dass auch über zwei asynchrone Nachrichten ein asynchrones Request-Response Message Exchange Pattern konstruiert werden kann. Wie noch zu sehen sein wird, übernimmt dieses Pattern eine fundamentale Rolle im Zusammenspiel zwischen Verbundanwendung und Servicevertrag-Implementierungsschicht.

Für Verbundanwendungen sind für die externe Kommunikation mit der Vermittlungsschicht sowohl synchrone als auch asynchrone Aufrufe zu verwenden. Die Unterscheidung richtet sich nach dem Anwendungsfall: für lesende Aufrufe aus Benut-

zeroberflächen heraus ist die synchrone Kommunikation zu bevorzugen, während schreibende Aufrufe grundsätzliche asynchron zu erfolgen haben. Für den Schreibvorgang wird ein spezielles Pattern verwendet, auf das noch in Kapitel 3.3.2.3 im Detail einzugehen sein wird. Die Motivation für die Benutzung synchroner Leseaufrufe aus Benutzeroberflächen heraus liegt in den Antwortzeiten begründet: der Endanwender wird schlechte Reaktionszeiten nicht akzeptieren. Das System muss schnellstmöglich auf Benutzeranfragen reagieren. Da die asynchrone Kommunikation zusätzlichen Aufwand/Overhead verursacht (Zuordnung der Antwort zu der Anfrage, Herstellung des Kontextes zu dem Zeitpunkt, als die Nachricht verschickt wurde usw.) und dadurch die Latenzzeiten erhöht werden, ist diese Kommunikationsform für schnelle Reaktionszeiten ungeeignet. Auch bei Verwendung eines synchronen Aufrufs sind kurze Antwortzeiten nicht garantiert. Schließlich definiert die Composite als Teil ihres Servicevertrags lediglich, welche Daten sie von der Implementierungsschicht verlangt. Unter Umständen müssen diese Informationen von der Implementierungsschicht für eine konkrete IT-Landschaft durch Aufrufe an mehrere Systeme erst eingesammelt werden. In diesem Fall entstehen Verzögerungen allein durch die Vielzahl der zu involvierenden Backends. In derartigen Situationen bleibt als Lösung lediglich das Replizieren der Daten in die Servicevertrag-Implementierungsschicht. Dann müssen zusätzliche Maßnahmen ergriffen werden, um die unterschiedlichen Datenbestände in Backend und Servicevertrag-Implementierungsschicht synchron zu halten. Hier haben sich beispielsweise MDM-Lösungen (Master Data Management) bewährt.

Aus dem Kapitel über die Spezifikation von Verbundanwendungen ist bekannt, dass Serviceaufrufe sowohl aus Benutzeroberflächen, aber eben auch aus Prozessschritten heraus erfolgen können. Dementsprechend kann für letztgenannte Lesezugriffe ebenfalls das asynchrone Verfahren verwendet werden, da kein Endanwender konkret auf eine Antwort wartet und im Fehlerfalle flexibler reagiert werden kann. Hier gilt es, den Aufwand zur Implementierung einer asynchronen Lösung abzuwägen, da diesbezüglich mehr Zeit (und damit Geld) im Vergleich zu einer synchronen Lösung zu investieren ist.

Diese Überlegungen gelten nicht für externe Schreiboperationen aus der Verbundanwendung heraus. Sie sollten, wenn möglich, grundsätzlich nicht aus Benutzeroberflächen heraus erfolgen. Der Grund ist in den Annahmen für eine Composite Application zu finden: da nicht bekannt ist, gegen welche konkrete IT-Landschaft die Compo-

site zum Einsatz kommt, ist auch deren Reaktionszeit unbekannt. Von daher sollten derartige Schreibzugriffe aus Prozessschritten heraus unter Verwendung des asynchronen Verfahrens erfolgen. Wie dies im Detail aussieht, wird Gegenstand des Kapitels 3.3.2.3 sein.

Zusammenfassend gilt für die Kommunikation der Verbundanwendung mit ihrer Servicevertrag-Implementierungsschicht:

- synchroner Kommunikationsstil bevorzugt für lesende Zugriffe aus Benutzeroberflächen heraus
- asynchrone Kommunikation für Schreiboperationen aus Prozessschritten heraus
- synchrone/asynchrone Kommunikation für lesende Zugriffe aus Prozessschritten heraus, je nach Anforderung und Aufwand
- Vermeidung asynchroner Schreiboperation aus Benutzeroberflächen heraus.

#### 3.3.1.3 Datenmodell/Typsystem

Das Datenmodell und damit einhergehend das verwendete Typsystem einer Verbundanwendung wurde ausführlich in Kapitel 3.2.7 behandelt. Wichtig ist die Verwendung eines kanonischen Datenmodells sowohl innerhalb einer Composite als auch zur Definition des Servicevertrags. Dies muss allerdings nicht zwangsläufig dazu führen, dass ausschließlich die einfachsten gemeinsamen Datentypen zu verwenden sind und es damit zu einer relativ schwachen Typüberprüfung kommt, wie dies Tabelle 5 suggeriert. Gerade durch die Wiederverwendung etablierter Standards (CCTS, SWIFT,…) lassen sich Datenmodelle entwerfen, die sowohl fachlich wichtige Datentypen umfassen, seitens ihrer Definition stabil sind und dabei gleichzeitig genau die Entkopplung von konkreten Backend-Systemen gewährleisten, die Verbundanwendungen benötigen. Die direkte Wiederverwendung existierender Datentypen aus Altsystemen in Composites ist hingegen zu vermeiden, da dadurch wieder Abhängigkeiten entstehen.

#### 3.3.1.4 Binding

Binding verfolgt die Frage, zu welchem Zeitpunkt die Zuordnungen von Symbolen/Aufrufen zu ihren Implementierungen erfolgen (Josuttis 2008, S. 57/58). Mögliche Zeitpunkte sind zur Kompilierzeit, zur Startzeit oder zur Laufzeit. Für Verbundanwendungen ist lediglich die Verbindung von Servicevertrag zur Implementierungsschicht von Interesse. Hier gilt: jeder Schnittstelle des Servicevertrags muss für die je-

weilige Ziellandschaft eine konkrete Implementierung zur Verfügung gestellt werden. Von daher ist die Verknüpfung von Servicevertrag und Implementierung statisch festgelegt, sie findet sich also nicht dynamisch zur Laufzeit. Sie lässt sich aber jederzeit zur Laufzeit austauschen, da es zu jeder potenziellen Schnittstelle mehrere Implementierungen geben kann, je nachdem, wie sich die Systemlandschaft verändert.

#### 3.3.1.5 Plattformspezifika

Verbundanwendungen nutzen selbst keinerlei Plattformspezifika der zu integrierenden Systeme aus. Dies bleibt der Servicevertrag-Implementierungsschicht überlassen, die die technischen Details kapselt. Dabei sollte sie durchaus Besonderheiten der jeweiligen Zielplattform ausnutzen, um das Zielsystem optimal zu nutzen und dadurch beispielsweise Performanz- oder Zuverlässigkeitsverbesserungen zu erzielen.

#### 3.3.1.6 Interaktionsmuster

Im Gegensatz zum Datenmodell, das bereits diskutiert wurde, wird mit „Interaktionsmuster“ die Komplexität der Service-Signatur adressiert, ob also eine Schnittstelle tief verschachtelt sein kann, ob sie Aufzählungstypen unterstützt oder ob Wertebereiche für einzelne Schnittstellenfelder verwendet werden können (Josuttis 2008, S. 58). Der Vorteil einer solchen komfortableren Schnittstelle liegt eindeutig in einer optimaleren Programmierung gegen derartige Schnittstellen: die zusätzlichen Informationen können verwendet werden, um frühzeitig Fehlerfälle zu erkennen. So werden bei Aufzählungstypen an der Benutzeroberfläche nur die möglichen Optionen angezeigt, die für dieses Feld vorgesehen sind. Fehleingaben sind dadurch ausgeschlossen. Genauso können Wertebereiche bereits bei der Eingabe verifiziert werden. Das Ergebnis ist eine wesentlich robustere Anwendung, die insbesondere in ihrem Kommunikationsverhalten weniger fehleranfällig ist, da mögliche Fehlerquellen frühzeitig eliminiert wurden.

Dem gegenüber steht einmal mehr eine erhöhte Abhängigkeit, falls die o.g. Signaturoptimierungen direkt den angeschlossenen Systemen entnommen wurden. Für Verbundanwendungen stellt sich diese Problematik aufgrund der Servicevertrag-Implementierungsschicht nicht, die einmal mehr derartige Details abstrahiert. Aufgrund der Ergebnisse der Diskussion des kanonischen Datenmodells, nämlich der Verwendung ein und desselben Datenmodells sowohl für die Verbundanwendung als auch für die Schnittstelle zur Servicevertrag-Implementierungsschicht, ergibt sich für die Kommunikation zwischen diesen beiden Partnern zwangsläufig die Verwendung von komple-

xen Strukturen. Dadurch kann auf Composite-Ebene wesentlich abstrahierter entwickelt und den Endanwendern deutlich mehr Komfort zur Verfügung gestellt werden. Auch bei den Verbindungen von der Implementierungsschicht zu den angeschlossenen Systemen muss ebenfalls auf die Ausnutzung plattformspezifischer Vorteile nicht verzichtet werden. Auf diese Weise werden sämtliche Vorteile kombiniert: der Anwender erhält benutzerfreundliche Oberflächen bei gleichzeitiger Nutzung plattformspezifischer Eigenschaften.

#### 3.3.1.7 Transaktionssicherheit

Die Transaktionssicherheit in lose gekoppelten Umgebungen stellt besondere Anforderungen an die Architektur von Verbundanwendungen, da das klassische „Two-Phase-Commit"-Protokoll (oder auch kurz „2PC") in einer verteilten Servicelandschaft versagt (siehe auch Golega 2007). Neben dem erhöhten Programmieraufwand, der Bereitstellung eines zentralen Ressourcen-Managers, sowie der Voraussetzung, dass sämtliche beteiligten Systeme dasselbe Transaktionsprotokoll unterstützen müssen, lässt ein 2PC als Lösung für verteilte Landschaften ausscheiden: damit 2PC funktionieren kann, müssen die beteiligten Systeme stets online sein. Diese Voraussetzung ist für Verbundanwendungen nicht erfüllt. Gerade in einem derart heterogenen Umfeld, in dem Composites einzusetzen sind, wird es immer Ausfälle von Systemen geben – sei es durch regelmäßige Software-Updates bzw. Wartung der Hardware oder durch auftretende Fehler wie Netzproblemen oder Systemabstürzen. Als Lösung bleibt lediglich die Kompensation, also das Vorsehen von Services, die im Falle eines Rollback einen bereits ausgeführten und aus Sicht einer Transaktion comitteten Service betriebswirtschaftlich zurückrollen kann. Daraus folgt für die Planung von Diensten, dass für verändernde Services stets eine kompensierende Lösung berücksichtigt werden muss. Darauf wird in einem späteren Kapitel noch im Detail einzugehen sein (Kapitel 4.1.5).

Für eine Verbundanwendung hat dies nun die folgenden unmittelbaren Konsequenzen: da die Composite nicht im Voraus bestimmen kann, wie lang eine verändernde Operation dauern wird, setzt sie schreibende Zugriffe aus der Prozessschicht in Form eines Hintergrundschrittes asynchron an die Servicevertrag-Implementierungsschicht ab, wartet anschließend also nicht unmittelbar auf eine Quittierung, sondern versetzt sich selbst nach dem Absetzen des Aufrufs in einen Wartezustand. Die Implementierungsschicht hat nun Gelegenheit, die geforderte betriebswirtschaftliche Funktionali-

tät ohne belastende Antwortzeitanforderungen abzuwickeln, da in der Composite selbst kein Anwender aktiv an einer Benutzeroberfläche wartet. Die Servicevertrag-Implementierungsschicht wiederum ruft die Backend-Systeme auf und kann im Fehlerfalle entsprechende Folgeaktivitäten einleiten. Dies können kompensierende Aufrufe sein, es kann sich dabei aber auch einfach nur um die Benachrichtigung von Administratoren (bei technischen Problemen) bzw. Fachexperten (bei betriebswirtschaftlichen Fehlern) handeln. Diese können dann das Problem durch spezielle Eingriffe lösen. An dieser Stelle soll es zunächst bei der Skizzierung der Problemlösung bleiben. In den Kapiteln 4.1.3 (Ausnahmebehandlung) und Kapitel 4.1.5 (Transaktionen und Kompensation) werden diese Pattern im Detail betrachtet.

#### 3.3.1.8 Kontrolle fachliche Logik

Bei der Kontrolle fachlicher Logik geht es darum, ob der Prozessfluss von einer zentralen Einheit, beispielsweise einer Prozess-Engine, heraus koordiniert wird oder ob die beteiligten Systeme sich durch den Versand von Nachrichten (beispielsweise über Zustandsänderungen an Geschäftsobjekten) selbst koordinieren (Josuttis 2008, S. 60). Im letztgenannten Fall kann die Koordinierung über gerichtete Point-to-Point-Verbindungen (P2P) erfolgen, oder über den Einsatz von Publish & Subscribe-Verfahren. Das Publish & Subscribe-Verfahren ist dabei aufgrund dessen Robustheit zu bevorzugen. Im Hinblick auf lose Kopplung ist das dezentrale Vorgehen robuster, während eine zentrale Prozess-Engine zu einem Engpass zur Laufzeit führen kann und sich Abhängigkeiten dadurch erhöhen.

Für Verbundanwendungen ergibt sich zu diesem Aspekt ein klares Bild: innerhalb der Composite werden die betriebswirtschaftlichen Abläufe in Form von konkreten Prozessmodellen umgesetzt, die anschließend von einer Prozess-Engine ausgeführt werden. Beim Übergang von der Composite zur Servicevertrag-Implementierungsschicht wird von der fachlichen zur technischen Seite gewechselt. Doch auch hier wird der technische Prozess in Form eines ausführbaren Modells umgesetzt. In dieser Beziehung sind Verbundanwendungen also eher enger gekoppelt, im konkreten Fall an die Ausführungs-Engine. Dies hat allerdings auch den Vorteil, dass der Prozessablauf konkret anhand der Modelle nachvollzogen werden kann und Fehler dadurch einfacher zu analysieren sind. Im dezentralen Szenario fehlen derartige Kontextinformationen und erschweren dadurch die Fehleranalyse.

#### 3.3.1.9 Versionierung

Ein wesentliches Merkmal von Verbundanwendungen ist deren Unabhängigkeit zu den konkreten Versionen der beteiligten Systeme. Deshalb wurde bewusst auf eine betriebswirtschaftlich ausgerichtete Definition des Servicevertrages unter Verwendung eines kanonischen Datenmodells Wert gelegt. Schließlich sollen die einzelnen Lösungen mit unterschiedlichen Tempo weiterentwickelt werden können: die Composite Application typischerweise in kürzeren, die Altsysteme in längeren Zyklen. Von daher sind Composites gänzlich lose gekoppelt in Bezug auf die Versionen der angeschlossenen Backend-Systeme. Erreicht wird dies durch die Abstraktion über die Servicevertrag-Implementierungsschicht. Sie sorgt für die jeweilig Adaption an die durch einen Versionswechsel bedingten Veränderungen. Die Verbundanwendung selbst bleibt davon gänzlich unberührt.

Der Vergleich der verschiedensten Aspekte einer losen Kopplung zwischen SOA, so wie sie in Tabelle 5 zusammengefasst sind, und denen einer Verbundanwendung bringt doch einige Unterschiede hervor. Von daher sind Verbundanwendungen als eine eigenständige Klasse von Anwendungen anzusehen, die sich von reinen SOA-Applikationen unterscheiden. Durch die Diskussion verschiedenster Aspekte der losen Kopplung ist zudem verdeutlicht worden, dass dieses Problem mehrdimensional zu betrachten ist. Für all die genannten Kriterien müssen bei der konkreten Composite-Entwicklung Lösungen gefunden werden. Dabei übernimmt die Servicevertrag-Implementierungsschicht die Hauptlast der Abstraktion. Dies verdeutlicht allerdings auch, dass für eine lose Kopplung, egal auf welcher Ebene sie zum Einsatz kommt, ein Preis in Form einer erhöhten Komplexität zu zahlen ist (Stiehl 2009e). Doch die gewonnenen Vorteile überwiegen die Nachteile: erst durch eine solche Architektur kann die Flexibilität und Agilität erzielt werden, die eine schnelle Anpassbarkeit an betriebswirtschaftliche Anforderungen aber auch an sich ändernde Systemlandschaften ermöglicht. Sie ist die Voraussetzung für eine zeitnahe Umsetzung der von der Unternehmensleitung vorgegebenen Strategie und somit letztendlich die Umsetzung des Business-IT-Alignments.

Aufgrund der vorangegangenen Diskussion verschiedenster Aspekte der losen Kopplung kann jetzt auch der Frage nach der Messbarkeit von „loser Kopplung“ (Frotscher 2009) nachgegangen und so ein Grad für die Abhängigkeit einer Anwen-

dung ermittelt werden. Eingangs wurde bereits darauf hingewiesen, dass es bei der losen Kopplung um die Reduzierung genau dieser Abhängigkeiten geht. Mit Tabelle 5 sind nun Kriterien gegeben, anhand derer eine Anwendung hinsichtlich ihrer Eigenschaft zur losen Kopplung überprüft werden kann. Für jedes Kriterium ist letztendlich die Frage zu beantworten, welche Auswirkungen eine Änderung genau dieses Kriteriums in einem der angeschlossenen Systeme auf die Verbundanwendung hat. Hat sie auch Eingriffe in der Composite zur Folge, ist bezüglich dieses Kriteriums eine enge Kopplung entstanden, andernfalls liegt bereits eine lose Kopplung vor. Als Beispiel sei das Kriterium „Physische Verbindung“ zur Verdeutlichung herangezogen. Die Frage lautet also: welche Auswirkungen hat die Veränderung der physischen Zieladresse oder auch die Veränderung des verwendeten Übertragungsprotokolls auf die Composite? Je nach Beantwortung dieser Frage können auf diese Weise mögliche Schwachpunkte erkannt und behoben werden. Ziel sollte es sein, den Grad der Abhängigkeit und damit die Anzahl der Annahmen über die involvierten Systeme auf ein Minimum zu reduzieren.

In diesem Zusammenhang ist es auch interessant, einen Blick auf das sogenannte SOA-Manifest (SOA Manifesto (Erl et. al. 2009)) zu werfen, in dem führende Autoren im SOA-Umfeld zusammengetragen haben, warum SOA im Allgemeinen gemacht wird, in welchem Kontext SOA sinnvoll ist und welche Richtlinien bei der Umsetzung von SOA zu beachten sind (Josuttis 2010). Das SOA-Manifest setzt sich aus einer Präambel, den Werte-Aussagen und Prinzipien zusammen. In der Präambel werden im Wesentlichen die beiden Begriffe *Service Orientierung* und *Service-orientierte Architektur* eingeführt und das Ziel formuliert, das mit SOA grundsätzlich verfolgt wird (nachhaltigen Geschäftswert liefern, höhere Agilität, Kosteneffizienz, Einklang mit sich ändernden fachlichen Bedürfnissen). Das Wertesystem behandelt die Priorisierung von sechs Werten, die bei SOA eine Rolle spielen, wie z.B. *Geschäftswert über technische Strategie* oder *Flexibilität über Optimierung*. Abgerundet wird das Manifest schließlich durch Prinzipien, die die Ziele von SOA und das dazugehörige Wertesystem untermauern. Dabei heißt es in einem der 14 Prinzipien wörtlich (Josuttis 2010, S. 26): „Reduce implicit dependencies and publish all external dependencies to increase robustness and reduce the impact of change.” Einmal mehr wird also die Bedeutung der Abhängigkeitsreduzierung hervorgehoben und als zentraler Bestandteil

einer Architektur für Anwendungen, die sich über mehrere Systeme oder ganze Systemlandschaften erstrecken, aufgeführt.

### *3.3.2 Aufgabenteilung und Zusammenspiel zwischen Verbundanwendung und Servicevertrag-Implementierungsschicht*

Nach diesen Vorarbeiten kann nun detaillierter auf die beiden Schichten „Verbundanwendung“ und „Servicevertrag-Implementierungsschicht“ eingegangen werden. Im Folgenden werden daher die Aufgabenteilung und das Zusammenspiel zwischen diesen beiden Schichten näher betrachtet.

#### 3.3.2.1 Verbundanwendung (Business Composition): Fokus auf benutzerzentrische Prozesse

Aufgrund der bisher geleisteten Arbeit können hinsichtlich der Architektur der Verbundanwendung selbst folgende wesentlichen Kernpunkte zusammengefasst werden (siehe auch Abbildung 9):

- Verbundanwendungen folgen einem Schichtenmodell bestehend aus den eigentlichen Geschäftsprozessen, den in den Prozessen verwendeten Benutzeroberflächen, sowie den für die Composite relevanten Geschäftsobjekten und Diensten
- Verbundanwendungen arbeiten durchgängig, also durch alle Schichten hindurch, ausschließlich auf Basis eines kanonischen Datenmodells
- Verbundanwendungen definieren extern benötigte betriebswirtschaftliche Funktionalität anhand eines Servicevertrags, der mittels WSDL ausgedrückt wird, der ebenfalls auf dem verwendeten kanonischen Datenmodell basiert und der eine klar definierte betriebswirtschaftliche Funktionalität fordert
- Prozesse, Benutzeroberflächen sowie Dienste kommunizieren ausschließlich über den Servicevertrag (und damit über die Servicevertrag-Implementierungsschicht) mit den Backend-Systemen. Kein Aufruf aus einer Verbundanwendung heraus umgeht diese Schicht

Die Ausprägungen der Prozesse, Oberflächen, Geschäftsobjekte und Dienste sind dabei ausschließlich betriebswirtschaftlich motiviert. Es handelt sich also um eine „Business Compositon“ mit dem Fokus auf benutzerzentrischen Prozessen (human-

centric processes). Dementsprechend sind die Benutzeroberflächen dabei rollenspezifisch ausgeprägt und unterstützen den Prozessbeteiligten in seiner jeweiligen Funktion und Prozesssituation optimal, d.h. sämtliche Informationen werden ihm kontextsensitiv zur Verfügung gestellt. Es handelt sich also nicht um allgemeingültige User Interfaces (UIs), sondern um spezifische, der Rolle und Situation entsprechend angepasste Oberflächen.

Die Verwendung des Begriffes *Composition* bringt dabei die Erbringung einer Leistung durch die Wiederverwendung existierender Dienste in Form von Prozessen zum Ausdruck. Bei der *Business Composition* wird folglich eine betriebswirtschaftlich-motivierte Dienstleistung erbracht, indem Benutzeroberflächen und Dienste zu neuen Lösungen komponiert werden.

Demselben Konzept der *Business Composition* wird auch auf der Geschäftsobjekt- und Serviceschicht gefolgt. Auch hier sind sowohl die Dienste als auch die Geschäftsobjekte ausschließlich dem Dienst der neuen Prozesse verpflichtet. Ihre Ausprägungen orientieren sich an dem fachlichen Bedarf. Für Geschäftsobjekte bedeutet dies eine Reduzierung der Attribute auf die in den Prozessen unbedingt notwendigen Felder. Die Dienste wiederum stellen für die Prozesse maßgeschneiderte Funktionalitäten zur Verfügung.

Sowohl für die Objekte als auch für die Dienste muss allerdings ein neuer Aspekt betrachtet werden, der in den anderen Schichten keine Rolle spielt: die Trennung zwischen internen und externen Objekten bzw. die Trennung zwischen intern und extern bereitgestellter Funktionalität.

Bei der Konzeption einer Verbundanwendung muss streng zwischen Objekten unterschieden werden, die vollständig neu sind und damit keine Repräsentationen in einem der angeschlossenen Systeme haben und Objekten, die aus den Backends wiederverwendet werden. Die beiden Objektarten sind dabei vollständig zu trennen, da sie auch unterschiedlichen Persistenzmechanismen folgen: für neue Objekte muss die Persistenz innerhalb der Verbundanwendung selbst erfolgen, d.h. Verbundanwendungen beinhalten demnach auch immer eine eigene lokale Persistenz für neue anwendungsspezifische Daten. Beispielsweise wird in einer neu zu entwickelnden Composite zur Behandlung und Weiterverarbeitung von innovativen Produktideen (siehe SAP's Lösung SAP xApp Product Definition – SAP 2002c) das Geschäftsobjekt „Idee“ zu definieren sein, das so sicherlich in keiner Standardanwendung zu finden sein wird.

Folglich muss die Speicherung dieses Objekts zwangsläufig in der Verbundanwendung selbst vorgenommen werden.

Im Gegensatz dazu benötigen die aus den extern bereitgestellten Systemen wiederverwendeten Objekte keine lokale Persistenz. Das Lesen der Objektdaten erfolgt dabei über einen Servicevertrag, der genau die für die Verbundanwendung wesentlichen Felder umfasst. Die Daten selbst werden dann der Composite in Form transienter Objekte (z.B. in Java oder C#) zur Verfügung gestellt. Dabei kann es durchaus passieren, dass die extern bereitgestellten Daten in Summe nicht den betriebswirtschaftlichen Anforderungen der Composite genügen. In solchen Fällen gilt auch hier: es handelt sich bei den noch fehlenden Informationen um neue, composite-spezifische Daten, die folglich wiederum in einem eigenen, lokal persistierten Objekt zu speichern sind. So können beispielsweise allgemeine Kundendaten aus einem CRM-System (Customer Relationship Management) in ein Objekt *CustomerCore* gelesen werden, die innerhalb der Composite allerdings noch zusätzlich benötigten Daten werden dann separat in dem lokal persistierten Objekt namens *CustomerExtension* abgelegt. Zwischen diesen beiden Objekten besteht zudem eine 1:1-Beziehung. Die explizite Trennung zwischen diesen beiden Objektarten ist allerdings nur durchführbar, wenn sie bereits zum Designzeitpunkt der Composite (siehe Kapitel zur Spezifikation von Verbundanwendungen) bekannt ist. Wird hingegen von den externen Applikationen erwartet, dass sie bestimmte Felder liefern sollten, die sie selbst allerdings nicht persistieren bzw. verwalten, so muss die entsprechende Funktionalität in der Servicevertrag-Implementierungsschicht nachgebildet werden. Wie dies genau umzusetzen ist, wird im unmittelbar nachfolgendem Abschnitt 3.3.2.2 erläutert.

Je nach Anwendungsfall und Peformanzanforderung kann bei extern beschafften Daten eine lokale Speicherung zwecks Caching ins Auge gefasst werden, um häufiges Nachlesen zu vermeiden. Dieses Verfahren eignet sich für Stammdaten recht gut, da sie sich in der Regel seltener ändern. Für transaktionale Daten stellt sich der Sachverhalt schon problematischer dar, da je nach Objekt die Änderungsrate recht hoch sein kann. In solchen Fällen muss situationsabhängig entschieden werden, welche Implementierung zu bevorzugen ist. Sollte die Entscheidung für ein Caching von Objektdaten fallen, so muss über die Notwendigkeit einer Datensynchronisation nachgedacht

werden. In derartigen Situationen sind dann klassische Caching-Algorithmen (siehe Hofmann 1984) einzusetzen.

Bei den Diensten verhält es sich ähnlich: auch hier wird zwischen neuer und extern wiederverwendeter Geschäftslogik unterschieden. Die neue Logik wird in eigenen, composite-spezifischen Diensten gekapselt und den höheren Schichten über standardisierte Schnittstellen zur Verfügung gestellt. An dieser Stelle steht eine mögliche spätere Wiederverwendbarkeit der Logik in anderen Lösungen zunächst nicht im Vordergrund. Von daher können die aus der SOA-Entwicklung bekannten Governance-Aspekte außer Acht gelassen werden, ein weiteres wichtiges Unterscheidungsmerkmal von Verbundanwendungen. Die Dienste sind ausschließlich nach den Bedürfnissen der Composite auszurichten. Dabei sollten bei der Gestaltung der Dienste allerdings Kriterien wie Abgeschlossenheit der zur Verfügung gestellten Funktionalität, Grobgranularität und Zustandslosigkeit des Dienstes durchaus berücksichtigt werden. Eine mögliche spätere Wiederverwendung der Dienste wird dadurch erleichtert.

Auch hinsichtlich der Aufrufe selbst ist die performanteste Variante zu wählen, da die Dienste lokal innerhalb der Verbundanwendung verwendet werden. SOAP-Zugriffe über HTTP sollten daher für die composite-interne Kommunikation vermieden werden. Lokale Methodenaufrufe in der jeweils verwendeten Programmiersprache sind daher das bevorzugte Mittel.

Die betriebswirtschaftlichen Funktionalitäten, die aus Sicht der Verbundanwendung extern zu erbringen sind, werden über den Servicevertrag definiert. Dieser basiert auf dem WSDL-Standard und verwendet ebenfalls das kanonische Datenmodell. Da sowohl die Verbundanwendung als auch die Servicevertrag-Implementierungsschicht unter der Kontrolle des Softwarelieferanten liegen und damit aus einem Hause stammen, können diese Schnittstellen sowohl komplexere Datenstrukturen als auch höherwertige Datentypen umfassen. Sind über die Schnittstelle komplette Objekte zu übertragen, die als solche bereits innerhalb der Composite verwendet werden, so ist die direkte Verwendung der zum Objekt gehörenden Datenstruktur innerhalb der WSDL zu berücksichtigen. Dieses Vorgehen hat dann Vorteile, wenn eine spätere modifikationsfreie Erweiterung der Composite durch den Software-Nutzer unterstützt werden soll. *Modifikationsfrei* bedeutet in diesem Kontext, dass der Kunde die Software derart erweitern kann, so dass...

1. ...der ursprüngliche Sourcecode für eine Erweiterung nicht angepasst werden muss und...
2. ...die vom Kunden eingebrachte Erweiterung auch nach einem Update der Ursprungslösung weiterhin aktiv bleibt. Nach dem Einspielen der Softwareaktualisierung wird die Kundenerweiterung automatisch wieder integriert.

Zur Umsetzung dieser Anforderungen wird in der Regel mit vom Softwarehersteller vordefinierten Erweiterungspunkten (extension points) gearbeitet. Dabei handelt es sich um Schnittstellen, die aus der Originalsoftware aufgerufen werden, sofern der Erweiterungspunkt aktiviert wurde. Wie ein solches Erweiterungskonzept für Verbundanwendungen aussehen kann und welche Rolle die Verwendung von Objekten dabei spielt, wird in einem späteren Kapitel ausführlicher diskutiert (Kapitel 6).

Aus technologischer Sicht sind für eine Verbundanwendung Entwicklungsumgebungen empfehlenswert, die eine integrierte Entwicklung von

- Prozessen
- Benutzeroberflächen
- Geschäftslogik
- Persistenz

ermöglichen. Derartige hochintegrierte Entwicklungssuiten haben sich insbesondere im Java-Umfeld etabliert. Hier sind natürlich die drei führenden Hersteller Oracle, IBM sowie SAP zu nennen. Auch Microsoft hat basierend auf C# eine entsprechende Umgebung im Programm. Auffällig ist die in den Suiten inzwischen unterstützte modellgetriebene Entwicklung. Es ist mittlerweile möglich, den Großteil einer Composite vollständig modellgetrieben zu erstellen: Prozesse werden mittels BPMN modelliert, die Benutzeroberflächen über WYSIWYG-Editoren (What You See Is What You Get) konstruiert und die in den Oberflächen angezeigten Daten mittels grafisch erstellter Klassendiagrammen umgesetzt. Selbst bei den Diensten können zumindest die Schnittstellen modelliert werden. Handelt es sich zudem bei einem konkreten Service um einen sogenannten Composed Service, der die Ergebnisse anderer Dienste aggre-

giert und diese als Ganzes zurückliefert, so kann auch diese Logik durch das Modellieren des Datenflusses zwischen den Service-Aufrufen ohne Programmierung realisiert werden. Die wirkliche Kodierungsarbeit reduziert sich somit deutlich und beschränkt sich ausschließlich auf die neue Geschäftslogik sowie der Behandlung von Spezialfällen und von Ausnahmesituationen, die so von den Modellierungstools nicht ohne Weiteres umgesetzt werden können. Von daher sollten Entwicklungsumgebungen APIs für die Fälle bereitstellen, die eine Erweiterung der Standardfunktionalität verlangen. In Summe steigt die Effizienz der Entwickler, die Umsetzungszeiten (und damit die Kosten) reduzieren sich und auch die Qualität kann verbessert werden.

Aufgrund der Fülle unterschiedlichster Artefakte einer Composite, zu deren Erstellung sowohl zur Entwicklungszeit aber eben auch zur Laufzeit Tools und Frameworks reibungslos zusammenspielen müssen, ist eine Best-of-Breed-Strategie nicht empfehlenswert. Neben der eigentlichen Implementierung der Verbundanwendung müssten dann noch zusätzliche Aufwände für die Integration der beteiligten Werkzeuge und Laufzeitumgebungen aufgrund diverser Unterschiede berücksichtigt werden. Zu diesen Unterschieden zählen:

- Unterschiedlichste, nicht aufeinander abgestimmter Programmiermodelle
- Unterschiedliche Fehlerbehandlung
- Unterschiedliche Datenmodelle
- Unterschiedliche Tools zur Fehleranalyse (kein einheitliches Logging)
- Unterschiedliche Sicherheitskonzepte
- Unterschiedliche Internationalisierungskonzepte
- Unterschiedliche Administration
- Unterschiedliches Monitoring
- Unterschiedliche Konfigurationskonzepte
- Unterschiedliche Benutzerverwaltungen
- Unterschiedliches Lebenszyklusverhalten der mit den Tools erstellten Komponenten
- Unterschiedlicher Support für Tools/Frameworks seitens der Hersteller

Letztendlich erhält der Entwickler bei der Best-of-Breed-Strategie keine ganzheitliche Sicht auf seine Verbundanwendung. Selbst bei den etablierten Herstellern fehlt dem Entwickler oft der Composite-Gesamtüberblick mit einer Darstellung aller Abhängigkeiten zwischen den entwickelten Komponenten. Wie so etwas aussehen könnte zeigt Abbildung 10.

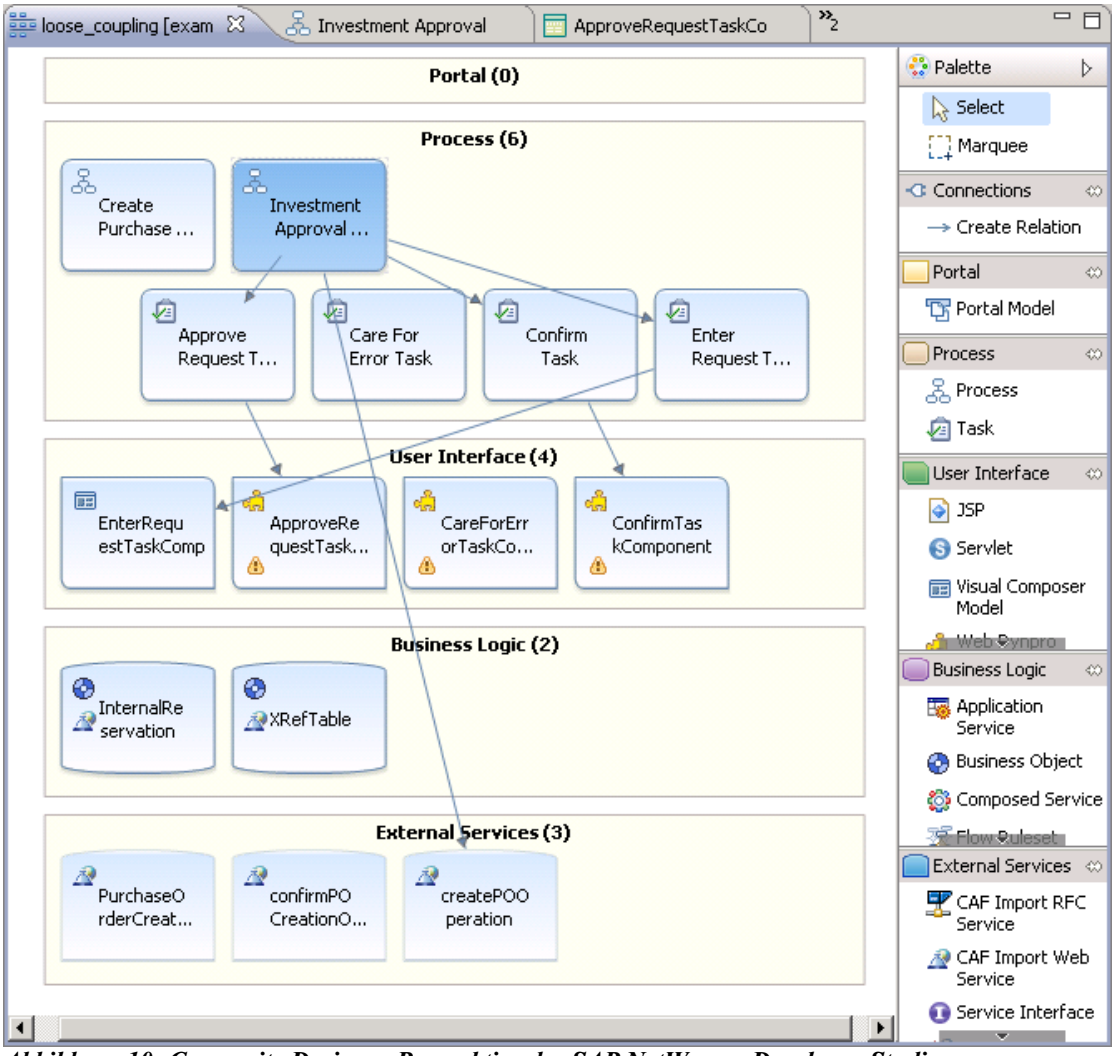



***Abbildung 10: Composite Designer Perspektive des SAP NetWeaver Developer Studios***

Sie zeigt einen Bildschirmabzug von SAP's sogenannter Composite Designer-Perspektive, die alle Artefakte einer Composite einschließlich ihrer Abhängigkeiten visualisiert. Gut zu erkennen ist auch die Schichteneinteilung der Composite, wie sie im Rahmen dieser Arbeit präsentiert wurde.

Schließlich darf ein weiteres großes Problem von Best-of-Breed nicht vergessen werden: die Überlebenschancen der an den Tools/Frameworks beteiligten Unternehmen und Open-Source-Lösungen. Gerade die immer beliebter werdenden Open-Sour-

ce-Projekte haben dabei mit der Gefahr zu Leben, dass höchst erfolgreiche Lösungen nicht mehr in dem Maße vorangetrieben werden, sobald der Hauptprotagonist des Projektes dieses nicht mehr leitet. Jüngstes Beispiel ist Tom Baeyens, der sich innerhalb der jBoss-Familie für die Geschäftsprozesskomponente jBPM verantwortlich zeichnete und kürzlich jBoss verließ, um sein neues Projekt Activiti (Baeyens 2010) voranzutreiben und damit jBPM mit einer ungewisse Zukunft zurücklässt. Diese Aspekte gilt es zu berücksichtigen, wenn es um diese doch recht wichtige und dabei gleichzeitig langfristig zu sehende Entscheidung um eine stabile Entwicklungsumgebung für Verbundanwendungen geht.

#### 3.3.2.2 Servicevertrag-Implementierungsschicht (Technical Composition): Fokus auf systemzentrische Prozesse

In der Servicevertrag-Implementierungsschicht ist der von der Verbundanwendung vorgegebene betriebswirtschaftlich motivierte Vertrag konkret gegen die vorhandene Systemlandschaft umzusetzen. Im Gegensatz zur Verbundanwendung ist die Implementierungsschicht hochgradig technologieabhängig: sie hat sich konkret mit den unterschiedlichsten Systemen und damit verbunden den unterschiedlichsten Datentypen, Protokollen, Sicherheitsmechanismen, Fehlerbehandlungen usw. auseinanderzusetzen und gegenüber der Composite Application zu abstrahieren und zu vereinheitlichen. Insbesondere die Datentypumsetzung (das Mapping) zwischen dem kanonischen Datentypsystem, das die Composite vorgegeben hat, und den jeweiligen Datentypsystemen der Backend-Systeme, gehört mit zu ihren Hauptaufgaben.

Die von der Composite verlangte betriebswirtschaftliche Logik kann nun von der Servicevertrag-Implementierungsschicht auf unterschiedlichste Weise erbracht werden. Im einfachsten Fall kann sie den Aufruf an genau ein Zielsystem weiterleiten, das exakt die geforderte Funktionalität liefern kann. In der Regel sind dann lediglich die obligatorischen Mappings (Struktur- und Datenmappings) zwischen den Schnittstellen vorzunehmen. Sollte bei der Kommunikation mit dem Backend-System ein Fehler auftreten, so muss je nach Kommunikationsart (synchron/asynchron) unterschiedlich reagiert werden:

- im synchronen Fall ist eine neutrale Fehlermeldung zu generieren, die den Endanwender über den Grund der nicht-lieferbaren Daten informiert. Zur Er-

innerung: die synchrone Kommunikation wird primär für lesende Aufrufe aus Benutzeroberflächen heraus initiiert. Da der Endanwender aber nichts über die hinter der Verbundanwendung agierende Systemlandschaft weiß, müssen Fehlermeldungen erzeugt werden, die von der konkreten Landschaft abstrahieren. Zudem ist zwischen betriebswirtschaftlichen und technischen Fehlergründen zu unterscheiden. Betriebswirtschaftliche Ursachen können dem Endanwender problemlos durchgereicht werden. Er kann anschließend seine Eingaben ändern und einen neuen Aufruf starten. Technische Probleme sind hingegen vom Endanwender fernzuhalten. Hier wird sich die Meldung auf eine sehr generische Information wie „Das System ist derzeit nicht erreichbar. Die zuständigen Administratoren wurden bereits informiert“ reduzieren, da ihm technische Details ohnehin nicht weiterhelfen würden. Nichtsdestotrotz muss er ein Feedback erhalten. Parallel dazu können bei technischen Problemen, je nach Fehlerursache, Systemmeldungen an Administratoren oder Fachabteilungen verschickt werden, die sich anschließend des Problems annehmen können.

– Im asynchronen Fall ändert sich die Situation signifikant: der Endanwender wartet nun nicht mehr aktiv auf eine Rückmeldung. Er hat seine Interaktionen mit der Verbundanwendung beendet, der betriebswirtschaftliche Prozess hat den Dialog abgeschlossenen, und in einem automatisierten Prozessschritt werden die Daten asynchron an die Servicevertrag-Implementierungsschicht zur Verbuchung weitergegeben. Unmittelbar nach der Übergabe legt sich der betriebswirtschaftliche Prozess schlafen und wartet nun auf seine Reaktivierung durch die Implementierungsschicht. Folglich müssen auftretende Fehler mit anderen Mechanismen gelöst werden, da der Kontakt zum Endanwender aufgrund der lose gekoppelten Implementierung verloren gegangen ist. Der Vorteil ist dennoch offensichtlich: durch die vollständige Entkopplung der Endanwenderaktivitäten von der eigentlichen Verbuchung in der Implementierungsschicht wird in einem gewissen Rahmen Zeit gewonnen. Es besteht nun nicht mehr die unbedingte Notwendigkeit, innerhalb weniger Sekunden antworten zu *müssen*, wie dies noch im synchronen Szenario der Fall war. Etwaigen Fehlern kann nun mit mehr Sorgfalt begegnet werden. Dennoch muss auch im asynchronen Fall zwischen technischen und betriebswirtschaftlichen Fehlern

unterschieden werden. Bei technischen Fehlern, wie sie typischerweise bei Netzausfällen, Systemabstürzen oder Ausfallzeiten aufgrund von Upgrade- bzw. Pflegearbeiten auftreten, werden entsprechende Nachrichten an die zuständigen Administratoren der betroffenen Systeme verschickt. Sie können sich nun zeitnah der Probleme annehmen. Da es sich um technische Probleme handelt, wird bei der asynchronen Kommunikation davon ausgegangen, dass das Problem aus Sicht der Verbundanwendung immer gelöst werden kann. Sobald die technischen Voraussetzungen wiederhergestellt sind, kann der Serviceaufruf erneut angetriggert und die betriebswirtschaftliche Funktionalität vervollständigt werden.

Anders verhält es sich bei betriebswirtschaftlichen Fehlern. Auch hier kann aufgrund der Entkopplung der Endanwender nicht mehr sofort zur Korrektur aufgefordert werden. Stattdessen müssen Fachexperten über die aufgetretene Situationen mittels Benachrichtigungen über Email oder in dringenden Fällen auch über SMS informiert werden. Diese haben nun die Möglichkeit, entweder selbstständig aufgrund ihrer Erfahrung und Fachkenntnis den Fehler zu beheben oder durch Kontaktaufnahme mit dem Endanwender eine Lösung herbeizuführen. Führt dies nicht zum erwünschten Erfolg, kann eine Fehlernachricht an den wartenden betriebswirtschaftlichen Prozess in der Verbundanwendung zurückgeschickt werden. Diese Fehlermeldung muss allerdings schon während der Definition des Servicekontrakts berücksichtigt worden sein, damit im weiteren Prozessverlauf auf Verbundanwendungsebene korrekt reagiert werden kann.

Die Komplexität der Umsetzung des Servicevertrags steigert sich, wenn mehr als ein System involviert ist. Dann kommt bei Schreiboperationen neben den bereits genannten Problemen zudem ein Transaktionsproblem hinzu. Es wird zunächst noch immer davon ausgegangen, dass die betriebswirtschaftliche Funktionalität vollständig durch Aufrufe an die Systemlandschaft erbracht werden kann. Es ist also keine zusätzliche Leistung in der Implementierungsschicht selbst zu realisieren. Nun muss geklärt werden, ob die Schreibaufrufe innerhalb einer Transaktion abzuwickeln sind oder nicht. Ist die Abwicklung innerhalb einer Transaktionsklammer notwendig, muss in den technischen Prozessen explizit das Zurückrollen von Transaktionen mit Hilfe

kompensierender Dienste berücksichtigt und ausmodelliert werden. Mögliche Fehlerbehandlungspattern werden dazu in Abschnitt 5.4 detailliert behandelt.

Als weitere Steigerung der Komplexität ist nun noch der Fall zu berücksichtigen, dass die verlangte fachliche Logik nicht mehr allein durch die Backend-Anwendungen geleistet werden kann. Es fehlen beispielsweise Felder, die so in den Backends nicht vorhanden sind oder es fehlen ganze betriebswirtschaftliche Funktionalitäten. In diesen Fällen ist diese Logik ebenfalls in der Servicevertrag-Implementierungsschicht umzusetzen. Dies lässt sich einfach aus den zuvor definierten Regeln zur Zusammenarbeit zwischen den beiden Schichten begründen: der Servicevertrag einschließlich der zu erbringenden Leistungen sind während der Spezifikationsphase eindeutig festgelegt worden. Darauf verlässt dich die Composite. Entsprechend muss die Implementierungsschicht diese Leistung auch vollständig liefern. Unter Umständen muss sie dazu eine eigene Persistenz für fehlende Felder umsetzen oder komplett neue Logiken implementieren. Wie immer die Anforderungen auch aussehen mögen, es ist bei der Implementierung ebenfalls nicht erlaubt, diese in eine der Endanwendungen vorzunehmen, da damit eine weitere wichtige Anforderung verletzt werden würde: die der Nicht-Invasivität. Als Lösung bleibt folglich einzig die Umsetzung in der Implementierungsschicht selbst, was allerdings auch sinnvoll ist, da dadurch eine Kapselung an einer zentralen Stelle gewährleistet wird, was die Wartung doch deutlich vereinfacht als wenn sie über verschiedenste Systeme verstreut wäre.

Wie diesen Ausführungen zu entnehmen ist, deckt die Servicevertrag-Implementierungsschicht komplexe, systemzentrische Prozessabläufe ab und kapselt dabei gleichzeitig sämtliche technischen Kommunikationsaspekte. Von daher wird diese Schicht auch als *Technical Composition* bezeichnet, die sich im Gegensatz zur benutzerzentrischen Business Composition auf Prozesse fokussiert, die die Verbindung zwischen den zu integrierenden Backend-Anwendungen herstellt. Analog zur Verwendung des Begriffes *Business Composition* wird bei der *Technical Composition* eine Dienstleistung durch Komposition existierender Funktionalität erbracht. Diese sind aber naturgemäß technischen Ursprungs.

In beiden Schichten stehen also immer Prozesse im Mittelpunkt. Folglich können bei einer späteren Implementierung für beide Schichten dieselben Vorgehensweisen

und Notationen verwendet werden. Details dazu werden in Kapitel 4 diskutiert, in dem es um die konkrete Umsetzung von Verbundanwendung und Servicevertrag-Implementierungsschicht gehen wird.

Zur Implementierung selbst können wiederum unterschiedlichste Technologien herangezogen werden. Wichtig ist lediglich die Erfüllung des Servicevertrags, so wie er in der Spezifikation festgelegt wurde. Dennoch empfehlen sich für die Implementierung asynchroner Aufrufe moderne Suiten, die den hohen Anforderungen an nichtfunktionalen Kriterien wie Zuverlässigkeit, Sicherheit, Robustheit, Skalierbarkeit sowie Performanz genügen. Zu derartigen Suiten zählen beispielsweise ESB-Implementierungen oder MOMs (Message Oriented Middleware). Moderne Lösungen erlauben mittlerweile auch die komfortable Modellierung asynchroner Abläufe entweder über das veraltete BPEL oder über das modernere BPMN. Innerhalb dieser Arbeit wird BPMN auch zur Implementierung der technischen Prozesse verwendet.

Für synchrone Aufrufe kann aus einem noch größeren Technologiefundus geschöpft werden. Egal ob Programmiersprachen wie Java, C#, C++ oder ABAP oder ob Scripting-Lösungen wie Ruby, Python oder Scala herangezogen werden: sie alle unterstützen die Implementierung von WSDL-Schnittstellen und sind daher für die Implementierung fachlicher Logik über synchrone Aufrufe prinzipiell erst einmal geeignet. Da sich die Servicevertrag-Implementierungsschicht allerdings in einem unternehmenskritischen Umfeld bewegt, müssen die bereits genannten nicht-fachlichen Kriterien bei der Auswahl unbedingt berücksichtigt werden. Von daher wird sich die Auswahl wieder reduzieren. Eine konkrete Empfehlung unterbleibt an dieser Stelle, da es für die Verbundanwendung primär um die Einhaltung des Servicevertrags geht – und die ist mit unterschiedlichsten Technologien zu erreichen.

In diesem und dem vorangegangenen Abschnitten wurden lediglich die Kernaufgaben von Verbundanwendung und Servicevertrag-Implementierungsschicht besprochen. In den folgenden Kapiteln und Abschnitten werden weitere Eigenschaften diskutiert, die von jeweils einer der beiden Seiten oder durch Kollaboration zu implementieren sein wird. Es wird in den jeweiligen Abschnitten erläutert, von wem welche Funktionalität konkret erbracht werden muss. Von daher ist die bisherige Betrachtung als generelle Sensibilisierung für die Aufgabentrennung zwischen den beiden Schichten zu verstehen, die sich wie folgt grob zusammenfassen lässt:

- Sämtliche fachlichen Aspekte gehören in die Verbundanwendung.
- Sämtliche technischen Aspekte im Zusammenspiel mit den zu integrierenden Endanwendungen gehören in die Servicevertrag-Implementierungsschicht.
- Die Verbundanwendung kommuniziert grundsätzlich über die Servicevertrag-Implementierungsschicht mit der IT-Landschaft. Ein Umgehen dieser Schicht ist nicht erlaubt.

Diese Definition kann bei der Entscheidungshilfe, was in welcher Schicht zu realisieren ist, als Orientierung dienen. Bereits der nächste Abschnitt beleuchtet die Zusammenarbeit der beiden Seiten anhand eines konkreten Szenarios und wird weitere Hilfestellungen bei der Entscheidungsfindung geben.

#### 3.3.2.3 Zusammenspiel zwischen Verbundanwendung und Servicevertrag-Implementierungsschicht

Doch wie arbeiten die beiden Schichten Verbundanwendung und Servicevertrag-Implementierungsschicht auch unter Berücksichtigung transaktionaler Aspekte konkret zusammen? Zur Veranschaulichung wird nun anhand eines konkreten Szenarios der Ablauf veranschaulicht und die Zusammenarbeit detailliert beschrieben. Das Szenario orientiert sich dabei im Wesentlichen an Stiehl 2010b. Insbesondere wird auf die bereits in Kapitel 3.3.1.2 (Kommunikationsstil) diskutierte Umsetzung der synchronen Kommunikation für Aufrufe aus Benutzeroberflächen sowie der asynchronen Kommunikation bei Schreiboperationen aus Prozessschritten heraus eingegangen. Als Beispiel wird ein vereinfachter Bestellprozess verwendet. Das dazugehörige BPMN-Diagramm veranschaulicht den Ablauf in Abbildung 11.

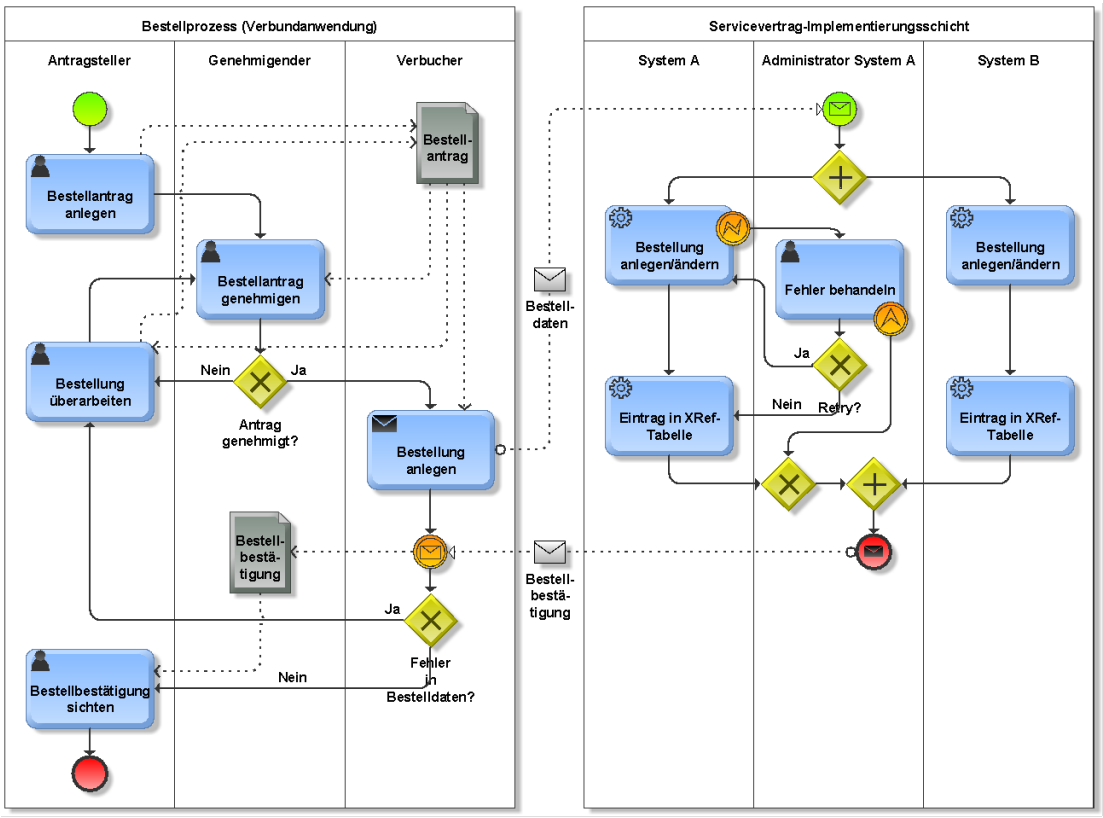



***Abbildung 11: Vereinfachter Bestellprozess als BPMN-Modell***

Die beiden Pools repräsentieren einerseits die Verbundanwendung selbst auf der linken Seite, andererseits die Servicevertrag-Implementierungsschicht auf der rechten Seite des Modells. Innerhalb der Verbundanwendung wird der eigentliche betriebswirtschaftliche Bestellprozess abgewickelt, innerhalb der Servicevertrag-Implementierungsschicht der dazu notwendige technische Prozess. Wie bereits im vorangegangenen Abschnitt ausgeführt wurde, können für Prozesse, unabhängig ob sie fachlicher oder technischer Natur sind, ein und dieselbe Notation verwendet werden. In unserem Fall ist es die BPMN. Sehr schön zu erkennen sind auch die unterschiedlichen Schwerpunkte der beiden Seiten: die Verbundanwendung ist eine benutzerzentrische Applikation, zu erkennen an der relativ häufigen Verwendung von Benutzeraufgaben innerhalb des BPMN-Modells. Auf der anderen Seite die systemzentrische Ausrichtung der Implementierungsschicht: bis auf eine Task handelt es sich bei allen anderen Schritten um automatisch abgewickelte Service-Aufgaben. Der Prozessablauf gestaltet sich nun wie folgt:

1. In der Verbundanwendung startet der fachliche Prozess über die Bearbeitung eines Antragsformulars für eine Bestellung durch den Antragsteller. Obwohl es in dem Modell nicht explizit aufgeführt wurde, so finden dennoch bereits zu diesem Zeitpunkt aus der dazugehörigen Benutzeroberfläche Aufrufe an die Servicevertrag-Implementierungsschicht statt: der Antragsteller wird zur Auswahl des zu bestellenden Produktes sicherlich in einem Katalog suchen wollen, der in der Regel in den Unternehmen schon vorhanden ist. Folglich wird während der Spezifikation eine Schnittstelle zu definieren sein, die sich an den konkreten Anforderungen der Composite orientiert und die extern zu erbringen sein wird. Beispielsweise genügt für dieses vereinfachte Szenario ein Suche, bei der als Suchfelder der Produktname und/oder die Produktnummer an einen Suchdienst übergeben werden. Als Rückgabe erwartet die Composite Application eine Liste von Produkten mit Produktnamen, Produktnummer sowie Preis. Aus dieser Trefferliste wird der Besteller anschließend das gewünschte Produkt auswählen, die erforderliche Bestellmenge ergänzen und den Bestellantrag absenden. Als Bestätigung des erfolgreichen Abschlusses seiner Aktivitäten erwartet der Antragsteller ein Bestätigung, z.B. in Form einer Bestätigungsnummer. Diese Nummer kann allerdings nicht von einem der Systeme aus der IT-Landschaft geliefert werden. Dadurch würde die Forderung verletzt, dass niemals asynchrone externe Schreibvorgänge aus einer Benutzeroberfläche heraus initiiert werden sollen. Dies macht auch deshalb keinen Sinn, da der Composite-Entwickler niemals wissen kann, in welchem konkreten Systemumfeld die Verbundanwendung eingesetzt wird. Wie wäre in einem solchen Fall zu verfahren, wenn gegen mehr als ein System verbucht werden müsste? Sollten dann sämtliche Nummern an den Besteller, der die im Hintergrund agierende Systemlandschaft nicht kennen kann, zurückgeliefert werden? Was könnte der Endanwender mit einer solchen Rückgabe anfangen? Außerdem wäre vollkommen unklar, wie lange ein solcher Aufruf dauern würde. Fraglich wäre auch das Verhalten im Fehlerfalle. Wie lange sollte man den Anwender dann warten lassen und welche Fehlermeldung ist dann zurückzugeben? Die Komplexität erreicht schnell nicht mehr handhabbare Größenordnungen.

Da dies alles keinen Sinn ergibt, muss eine Alternative gefunden werden, die sich wie folgt gestaltet: statt die Daten bereits zu diesem Zeitpunkt an die Servicevertrag-Implementierungsschicht und damit an die Backend-Systeme zu übergeben, werden die Antragsdaten *lokal* in der Verbundanwendung innerhalb eines eigens dafür vorgesehenen Geschäftsobjekts gespeichert und die dabei erzeugte eindeutige Identifikationsnummer an den Endanwender als Quittung zurückgeliefert. Da es sich um einen lokalen Service-Aufruf handelt, ist die Ausführung performant ausführbar mit entsprechend kurzer Wartezeit für den Antragsteller. Zur weiteren Erläuterung des Szenarios sei an dieser Stelle angenommen, dass bei der lokalen Speicherung der Antragsdaten die *interne* Nummer 4711 erzeugt wurde. Mit der Abspeicherung der Auftragsdaten und der Erzeugung der internen Nummer kann die erste Prozessaktivität abgeschlossen werden. Wie dem Prozessmodell entnommen werden kann, wird das prozessinterne Datenobjekt *Bestellantrag* erzeugt. Hierbei handelt es sich konkret um den Inhalt des bereits im Kapitel zur Spezifikation von Verbundanwendungen eingeführten Prozesskontextes. Die Daten des Prozesskontextes stehen allen Schritten des Prozesses zur Verfügung und sie leben so lange, wie die dazugehörige Prozessinstanz aktiv ist. Im Umkehrschluss bedeutet dies allerdings auch, dass auf die Daten des Prozesskontextes nach Prozessende nicht mehr zugegriffen werden kann. Wie noch zu sehen sein wird, erfolgt die Übergabe dieser Antragsdaten zur eigentlichen Verbuchung an die Implementierungsschicht zu einem späteren Zeitpunkt durch einen asynchronen Aufruf.
Der Antragsteller beendet jetzt jedenfalls seinen Schritt und initiiert damit die Genehmigung des Antrags durch die Rolle, die im Pool mit *Genehmigender* bezeichnet ist.

2. Der Genehmigende hat lediglich die vom Antragsteller eingegebenen Daten zu überprüfen und entweder zu genehmigen oder abzulehnen. Dazu erfolgt eine Datenübergabe aus dem Datenobjekt *Bestellantrag* heraus an den Dialogschritt des Genehmigenden. Hinsichtlich der Diskussion zur Zusammenarbeit zwischen Verbundanwendung und Implementierungsschicht liefert dieser Schritt keinen weiteren Beitrag. Nach Ablehnung des Antrags erhält

der Antragsteller die Möglichkeit, etwaige Korrekturen vorzunehmen. Dazu wird seine Benutzeroberfläche wiederum aus dem Objekt *Bestellantrag* heraus gefüllt und etwaige Änderungen wieder zurückgeliefert. Da der Antragsteller auch in diesem Schritt unter Umständen wieder suchen und die Daten lokal abspeichern muss, gilt das unter Punkt 1 Gesagte.
Im Falle einer Genehmigung können die Daten zur eigentlichen Verbuchung in der IT-Landschaft an die Servicevertrag-Implementierungsschicht übergeben werden. Damit beginnt nun die asynchrone Zusammenarbeit der beiden Schichten, die im Folgenden detailliert diskutiert wird.

3. Dem Modell aus Abbildung 11 folgend wird die Sendeaktivität *Bestellung anlegen* innerhalb der Verbundanwendung in der Bahn *Verbucher* erreicht. Sendeaktivitäten werden innerhalb der BPMN verwendet, um asynchrone Kommunikationsschritte mit anderen Prozessen darzustellen. Dies ist exakt, was auch innerhalb des Zusammenspiels zwischen Verbundanwendung und Servicevertrag-Implementierungsschicht zum Ausdruck gebracht werden soll: eine Nachricht wird an die *Technical Composition* übergeben, wobei der Inhalt der Nachricht dem Servicevertrag zwischen Composite und Implementierungsschicht entspricht. Der Vertrag für dieses Szenario könnte wie folgt aussehen:

```
<xsd:element name="createPurchaseOrderOperation">
<xsd:complexType>
 <xsd:sequence>
   <xsd:element name="correlationID type="xsd:string"/>
    <xsd:element name="productID" type="xsd:string"/>
    <xsd:element name="quantity" type="xsd:decimal"/>
 </xsd:sequence>
</xsd:complexType>
</xsd:element>
```

Der Vertrag sieht aus Sicht der Composite demnach lediglich die Übergabe des gewählten Produktes samt der Bestellmenge vor. Die Schnittstelle umfasst allerdings ein weiteres Feld namens *correlationID*. Über dieses Feld

kann später bei der Rückmeldung des Ergebnisses aus der Implementierungsschicht die dazugehörige Composite-Prozessinstanz gefunden werden, die ursprünglich den Verbuchungsauftrag initiiert hatte. Diese Zuordnung ist von daher wichtig, als dass mehrere Instanzen des Bestellprozesses parallel abgewickelt werden können. Dann muss die korrekte Zuordnung der Antwort zur Anfrage gewährleistet sein. Erreicht wird dies über ein eindeutiges Identifikationsmerkmal, in diesem Beispiel mit *correlationID* bezeichnet. Als möglicher Inhalt bietet sich die in der Composite bereits erzeugte lokale Auftragsnummer an.
Wie dem Modell zu entnehmen ist, werden die Schnittstellendaten ebenfalls mit den Inhalten des Prozesskontextes gefüllt. Nach der asynchronen Übergabe der Daten an die Implementierungsschicht erreicht der Prozess der Verbundanwendung schließlich das Nachrichten-Zwischenereignis, an dem er auf die Bestellbestätigung mit der passenden *correlationID* wartet. In der Literatur wird das Pattern zur Korrelation zweier Prozesse über eine eindeutige ID auch mit Correlation Identifier bezeichnet (siehe Hohpe & Woolf 2004, S. 163).

Eine interessante Variante des dargestellten Prozessablaufs stellt die Ausführung weiterer Aktivitäten parallel zum Wartevorgang dar, so wie es in der folgenden Abbildung 12 dargestellt ist:


***Abbildung 12: Parallel warten und weitere Aktivitäten ausführen***

So kann die Wartezeit mit anderen Aktivitäten genutzt werden. Sind sowohl die zusätzlichen Aktivitäten als auch die Quittierung der Bestellung eingetroffen (Gateway zur parallelen Zusammenführung), kann der Prozess mit der weiteren Bestellabwick-

lung fortgesetzt werden.

4. Was passiert innerhalb der Servicevertrag-Implementierungsschicht? In Abbildung 11 ist für diese Schicht nur der reine Sequenzfluss veranschaulicht worden – auf den Datenfluss wurde zwecks Übersichtlichkeit verzichtet.
   Nach Erhalt der Bestelldaten (Nachrichten-Startereignis) zweigt sich der Prozessfluss an dem parallelen Gateway auf, um zu veranschaulichen, dass zur Implementierung der fachlichen Anforderung *Bestellung verbuchen* zwei Systeme unabhängig voneinander involviert werden können. Für eine andere Systemlandschaft sähe die Implementierung natürlich vollkommen anders aus – aber genau hier liegt der Reiz in der Zusammenarbeit von Composite und Implementierungsschicht. Dank klarer und vollständiger Trennung sind vielfältige Umsetzungsmöglichkeiten der Composite-Anforderungen denkbar.

5. Sowohl für System A als auch für System B werden die Bestelldaten durch Service-Aufgaben verbucht. Dabei wird für System A zudem beispielhaft veranschaulicht, wie eine mögliche Fehlerbehandlung aussehen könnte. In produktiven Implementierungen müsste eine derartige Fehlerauswertung natürlich auch für System B erfolgen, allerdings wurde diese aus Übersichtlichkeitsgründen in Abbildung 11 nicht ausmodelliert. Die Fehlerbehandlung selbst wird nun durch das an die Service-Aktivität *Bestellung anlegen/ändern* für System A angeheftete unterbrechende Fehlerereignis zum Ausdruck gebracht: in einem solchen Fall wird dem Sequenzfluss gefolgt, der aus dem Fehlerereignis herausführt. Es wird folglich ein Administrator über den aufgetretenen Fehler zeitnah informiert. In dem Beispiel kann der Administrator entscheiden, ob der fehlerverursachende Service nochmals aufgerufen werden soll (Gateway *Retry?*) oder ob der Fehler bereits vom Administrator manuell durch einen entsprechenden Eingriff in dem Zielsystem behoben werden konnte und der Prozess unmittelbar mit dem darauffolgenden Schritt fortgesetzt werden soll. Je nach Entscheidung wird dem dazugehörigen Prozessfluss gefolgt. Natürlich hätte anstelle eines Administrators, der eher für technische Fehler zuständig ist, im Falle eines

fachlichen Fehlers ein Mitarbeiter der Fachabteilung hinzugezogen werden können. Der Sachbearbeiter hätte durch Eingaben im jeweiligen Zielsystem, gegebenenfalls auch unter Rücksprache mit dem Antragsteller, ebenfalls das Problem lösen können. Wichtig an dieser Stelle ist, dass sämtliche Fehlerfälle, egal welcher Couleur, dicht an den Backendsystemen behandelt und nicht einfach an die Verbundanwendung durchgereicht werden. Zu diesen frühen Zeitpunkten ist die Wahrscheinlichkeit, die Fehlerursache erkennen und damit korrekt beheben zu können, einfach am Größten. Zusammenfassend sind folglich sowohl *technische* als auch *systemspezifisch fachliche* Fehler innerhalb der Servicevertrag-Implementierungsschicht zu behandeln. *Systemspezifisch fachlich* bedeutet, dass der Fehler konkret mit dem verwendeten Zielsystem in Zusammenhang steht. Ist die Fehlerursache allerdings unabhängig vom Zielsystem, so handelt es sich um einen *systemneutralen fachlichen* Fehler, der an die Verbundanwendung zurückzugeben ist und dort behandelt werden muss. Angedeutet wird dies in der Abbildung durch den Systemfluss, der aus dem an den *Fehler behandeln*-Schritt des Administrators geheftete unterbrechende Eskalationsereignis herausführt. Der Administrator hat also in seinem Dialogschritt die Möglichkeit, systemneutrale fachliche Fehler, die er nicht behandeln kann, durch eine Eskalation anzuzeigen und für eine entsprechende Behandlung zu sorgen. In diesem Fall also die Rückgabe an die Verbundanwendung. Dies ist auch von daher sinnvoll, als dass derartige Fehler ohnehin nur innerhalb der Composite gelöst werden können.

6. Sowohl für System A als auch für System B wird nach Anlage der Bestellung in dem jeweiligen Zielsystem der Schritt *Eintrag in XRef-Tabelle* erreicht. In dieser Cross-Referenz-Tabelle werden Referenzen zwischen der internen Nummer der Composite und den jeweiligen Nummern der Zielsysteme verwaltet. Dazu wird in der Implementierungsschicht ein eigens dafür vorgesehenes lokal-persistiertes Objekt verwendet und gepflegt.
   Diese Verwaltung der Cross-Referenz-Tabelle muss eindeutig die Servicevertrag-Implementierungsschicht übernehmen, da die Tabelle technische Details der integrierten Endsysteme beinhaltet. Technische Details liegen

aber stets außerhalb der Verbundanwendung und damit in der Implementierungsschicht. Wie die Tabelle konkret für das Beispielszenario aussehen könnte, ist in Tabelle 6 dargestellt. Zur Befüllung der Tabelle wird für die ID der Verbundanwendung das Feld *correlationID* des Servicevertrags herangezogen, während für die IDs der Systeme A und B die entsprechenden Felder aus den Rückgabeparametern der jeweiligen *Bestellung anlegen/ändern*-Serviceaufrufe an die Systeme selbst genommen werden.

| ID der Verbundanwendung | ID System A | ID System B |
|---|---|---|
| 4711 | 815 | 4500002010 |

***Tabelle 6: Cross-Referenz-Tabelle für Beispielszenario***

Die Cross-Referenz-Tabelle ist folglich bei sämtlichen Aufrufen zukünftig zu berücksichtigen, unabhängig davon, ob es sich dabei um lesende oder schreibende Aufrufe handelt. Gibt es beispielsweise für das Szenario neben dem Bestellabwicklungsprozess auch noch einen Auftragsänderungsprozess, so müssen Daten aus den Systemen A und B nachgelesen werden, damit Änderungen vorgenommen werden können. Wird also, um bei den Beispieldaten zu bleiben, der Composite-Auftrag mit der ID 4711 geändert, müssen die Daten aus System A unter der Nummer 815 und aus System B unter der Nummer 4500002010 nachgelesen werden. Umgekehrt ist aus der Kenntnis der Nummer in den beteiligten Backend-Anwendungen unmittelbar ersichtlich, welches Objekt in der Composite von etwaigen Änderungen in den Backends betroffen ist. Dieses Wissen kann genutzt werden, um die Composite auf Basis von Ereignisse über diese Änderungen zu informieren und ihr dadurch Gelegenheit zu geben, passende Maßnahmen zu ergreifen. Die Einträge müssen dabei so lange in der Tabelle gehalten werden, wie auch die dazugehörigen Geschäftsobjekte noch nicht gelöscht sind. Dies gilt es insbesondere bei Archivierungsläufen zu berücksichtigen, die dann gleichzeitig für Verbundanwendung, Backend-System und Cross-Referenz-Tabelle durchzuführen sind. Wird eines der in der Cross-Referenz-Tabelle berücksichtigten Backend-Systeme durch das eines anderen Herstellers ersetzt oder entfällt es vollständig aufgrund von Konsolidierungsmaßnahmen im Unternehmen, so muss selbstverständlich

auch die Aktualisierung der Tabelle berücksichtigt werden. Bei einer Systemkonsolidierung ist die betroffene Spalte aus der Tabelle zu entfernen, während bei einem Systemwechsel die Identifikationsnummern des alten Systems gegen die des neuen Systems ersetzt werden müssen. Dies wird im Rahmen der Datenmigration von dem Alt- auf das Neusystem vorgenommen.
Die Cross-Referenz-Tabelle übernimmt folglich eine zentrale Rolle im Zusammenspiel zwischen Verbundanwendung und Servicevertrag-Implementierungsschicht und ist somit ein wichtiger Bestandteil der Implementierungsschicht.

7. Nach der erfolgreichen Aktualisierung der Cross-Referenz-Tabelle erfolgt das Aufwecken des wartenden Composite-Prozesses. Das parallele Gateway sorgt zunächst für die Synchronisation der beiden Pfade. Erst wenn beide erfolgreich beendet wurden, erfolgt die Fortsetzung mit dem sendenden Ende-Ereignis. Auch diese Nachricht ist Bestandteil des Servicevertrags und könnte wie folgt aussehen:

```
<xsd:element name="confirmPurchaseOrderOperation">
<xsd:complexType>
 <xsd:sequence>
   <xsd:element name="correlationID type="xsd:string"/>
    <xsd:element name="errorCode" type="xsd:string"/>
    <xsd:element name="errorMsg" type="xsd:decimal"/>
 </xsd:sequence>
</xsd:complexType>
</xsd:element>
```

   Offensichtlich wird keine der von den integrierten Systemen erzeugten Nummern an den Verbundanwendungsprozess zurückgeschickt. Diese Information wäre ohnehin nutzlos, da die Composite keine Annahmen über die Systemlandschaft macht. Die *correlationID* sorgt für die Identifikation des richtigen wartenden Prozesses. Nach Versand der Daten beendet sich der technische Prozess.

8. Die Bestellbestätigungsdaten werden auf Seiten der Verbundanwendung wiederum in den Prozesskontext geladen. Anschließend wird in einem exklusiven Gateway auf mögliche Fehlermeldungen der Implementierungsschicht reagiert. Durch dieses Vorgehen wird eine einfache *systemneutrale fachliche* Fehlerbehandlung auf Seiten der Verbundanwendung angedeutet, die im Gegensatz zur sowohl *technischen* als auch *systemspezifischen fachlichen* Fehlerbehandlung innerhalb der Implementierungsschicht steht. Im Falle eines solchen Fehlers hat der Antragsteller erneut die Möglichkeit, seine Daten entsprechend zu korrigieren und erneut zur Genehmigung vorzulegen.

   Konnte die Bestellung hingegen erfolgreich in den Systemen verbucht werden, so sind die zur weiteren Bearbeitung notwendigen Daten aus dem Prozesskontext heraus an die Oberfläche des Antragstellers zur Anzeige der Bestellbestätigung weiterzureichen. Nach Quittierung der Daten kann auch dieser Prozess letztendlich beendet werden.

Mit diesem Beispielszenario wurden die wesentlichen Aspekte der Grundarchitektur einer Verbundanwendung in Zusammenspiel mit der Servicevertrag-Implementierungsschicht veranschaulicht und wesentliche Aufgaben auf die beiden Schichten verteilt. Basierend auf diesem Grundgerüst werden im Folgenden weitere Details von Verbundanwendungen und Implementierungsschicht diskutiert, um auf der einen Seite die Trennung und damit die Aufgabenverteilung zwischen den beiden Ebenen weiter zu schärfen, aber auch um auf der anderen Seite die Architektur für den Einsatz in unternehmenskritischen Szenarien zu stabilisieren, sie robuster zu machen und dabei gleichzeitig deren Flexibilität zu erhöhen. Im nächsten Abschnitt wird dazu in einem ersten Schritt eine weitere Verfeinerung des Modells vorgenommen, indem bei der Umsetzung der Servicevertrag-Implementierungsschicht ein Enterprise Service Bus berücksichtigt wird.

#### 3.3.2.4 Berücksichtigung eines ESB's in der Servicevertrag-Implementierungsschicht

Typische Aufgaben eines Enterprise Service Bus (ESB) wie z.B. das Routen von Nachrichten, die Nachrichtentransformation durch Mapping sowie die Verwaltung der technischen Anbindungen an die involvierten Systeme, müssen nicht notwendigerweise in der Servicevertrag-Implementierungsschicht ausmodelliert werden. ESBs sind für derartige Aufgaben optimiert und entlasten den Entwickler deutlich von typischen Routinearbeiten bei der Nachrichtenvermittlung in heterogenen Umgebungen. Sie stellen daher eine vielversprechende Ergänzung in der Architektur von Verbundanwendungen dar. Doch welche konkreten Auswirkungen hat der Einsatz eines ESB's für Verbundanwendungen bzw. für die Servicevertrag-Implementierungsschicht? Da es sich bei den oben aufgeführten ESB-Aufgaben ausschließlich um technische Dienste handelt, ist die Verbundanwendung selbst nicht vom ESB-Einsatz betroffen. Nach wie vor stellt der Service-Vertrag ihre stabile Schnittstelle dar. Lediglich in der Implementierungsschicht müssen Aufgabentrennungen vorgenommen werden. So ist dessen Umsetzung, wie in Abbildung 11 dargestellt, noch immer anfällig für die Erweiterung bzw. Reduktion von Zielsystemen. Für jedes Backend-System, das entweder neu zu berücksichtigen ist oder durch Konsolidierungen entfällt, muss der Prozess am parallelen Gateway entsprechend angepasst werden. Auch in diesem Fall lässt sich durch eine Trennung von Aufgaben zwischen Integrationsprozess und ESB eine zusätzliche Stabilisierung erreichen, so dass derartige Änderungen keine Auswirkungen mehr auf den Integrationsprozess haben. Dazu zeigt die Abbildung 13 auf der nächsten Seite eine mögliche Lösung.

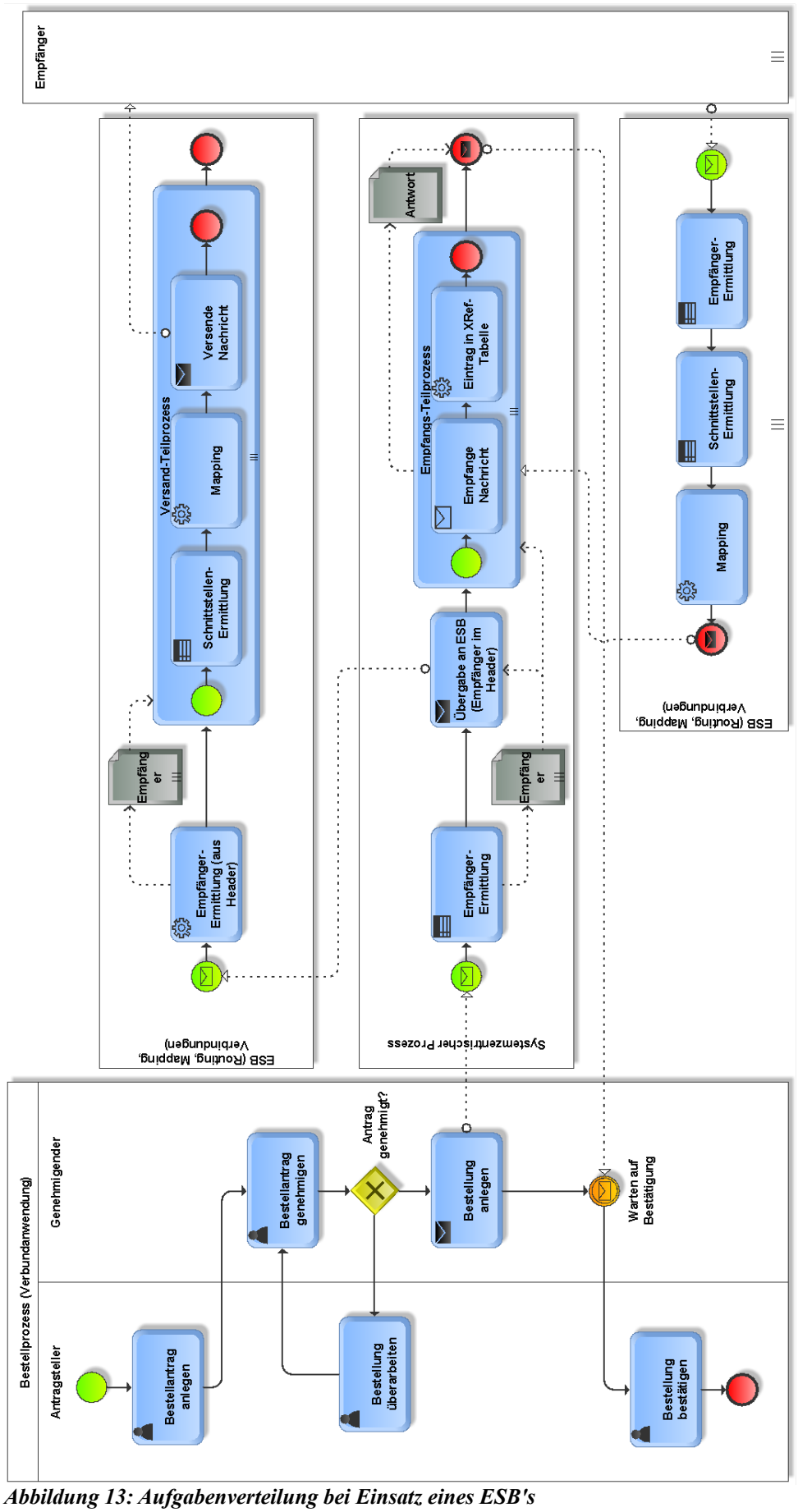



*Abbildung 13: Aufgabenverteilung bei Einsatz eines ESB's*

Auf der linken Seite ist die eigentliche Verbundanwendung zu erkennen, die, wie gewohnt, das Anlegen der Bestellung asynchron an die Implementierungsschicht in Form einer Nachricht übergibt. Beim Empfänger der Nachricht handelt es sich um einen systemzentrischen Prozess, der die Koordination der Aktualisierungsaufrufe an die beteiligten Backend-Systeme übernimmt. Nach Übergabe der Verantwortung an die Implementierungsschicht, wartet der Hauptprozess an dem nachrichtenbasierten Zwischenereignis auf die erfolgreiche Bearbeitung seines Auftrags. Der systemzentrische Prozess ermittelt zunächst über ein Regelwerk die Empfänger der Bestellung. Dabei greift er auf exakt dasselbe Regelwerk zurück, das auch der ESB bei seiner Routing-Ermittlung heranziehen würde. Diese Aufgabe übernimmt im aktuellen Fall allerdings der systemzentrische Prozess, da er später auf die Bestätigungen warten muss. Daher bestimmt er vorab, um wie viele Empfänger es sich dabei handelt. Der Prozess übernimmt selbst auch nicht das eigentliche Routing der Nachricht. Das bleibt nach wie vor Aufgabe des ESB's. Da der systemzentrische Prozess die Empfänger nun schon ermittelt hat, kann er diese in einem speziellen Header-Bereich der Nachricht hinterlegen, die er dem ESB übergibt. Die Sende-Aufgabe im systemzentrischen Prozess ist auch nicht als Schleife ausgelegt, auch wenn mehrere Empfänger ermittelt wurden. Nach Übergabe der Nachricht an den ESB wartet der Prozess in einem mehrfach instanziierten Empfangs-Teilprozess auf die Antworten. Die Anzahl der Instanzen entspricht dabei der Anzahl der ermittelten Empfänger, da von jedem System eine Antwort erwartet wird. Der Empfang ist deshalb als Teilprozess ausgelegt, da nach jeder empfangenen Nachricht noch ein Eintrag in der Cross-Referenz-Tabelle vorgenommen werden muss, was ebenfalls abhängig vom jeweiligen Backend-System ist. Die Korrelationsbedingung für den Empfangsschritt setzt sich aus der übergebenen ID für die lokal gespeicherten Antragsdaten der Verbundanwendung und der Kennung für das Zielsystem (konkret: der ermittelte Empfänger) zusammen. Auf diese Weise wird jede Instanz des Empfangs-Teilprozesses eindeutig identifiziert. Aufgrund dieses neuen, generischeren Ansatzes ist die ursprünglich vorgeschlagene Cross-Referenztabelle (Tabelle 6) nicht mehr geeignet, da sie für jedes involvierte System eine eigene Spalte vorsah und dadurch für Änderungen in der Systemlandschaft zu starr ist. Jedes neue System verlangt das Hinzufügen einer neuen Spalte bzw. jedes entfernte System eine entsprechende Spaltenlöschung. Da sich die Systemzusammenstellung jedoch jeder-

zeit ändern kann, muss sich dies auch in der Cross-Referenztabelle widerspiegeln. Sie hat nun das in Tabelle 7 gezeigte Aussehen:

| ID der Antragsdaten in der Verbundanwendung | ID des Zielsystems | ID des Auftrags im Zielsystem |
|---|---|---|
| 4711 | System A | 4500002010 |
| 4711 | System B | 815 |
| 4711 | ERP | 1GWL972 |
| | | |

***Tabelle 7: Flexible Cross-Referenztabelle***

Jetzt findet in jeder Tabellenzeile eine Zuordnung von der internen ID der Verbundanwendung zu der ID im Zielsystem statt, wobei das jeweilige Zielsystem nun ebenfalls explizit in der Tabelle mitaufgenommen wurde (mittlere Spalte). Die ersten beiden Spalten reflektieren zudem die Korrelationsbedingung für den Empfangsschritt, von dem bereits weiter oben die Rede war. Diese Tabelle ist nun für jegliche Änderungen in der IT-Landschaft gerüstet und unterstützt die durch die Gesamtarchitektur angestrebte Flexibilitätssteigerung.

Da der systemzentrische Prozess den Servicevertrag implementiert und die darin verwendeten Datentypen auf dem kanonischen Datenmodell beruhen, ist es naheliegend, dass auch im gesamten systemzentrischen Prozess ausschließlich auf dem kanonische Datenmodell gearbeitet wird. Das macht auch von daher Sinn, als dass der Prozess mit den proprietären Datentypen der Backend-Systeme nicht in Kontakt kommt. Dies bleibt einmal mehr dem ESB überlassen.

Der ESB entnimmt zunächst in der Service-Aufgabe *Empfänger-Ermittlung* die Liste der Nachrichtenempfänger, die zuvor vom systemzentrischen Prozess bestimmt wurden. Gemäß der gefundenen Empfängeranzahl wird ein mehrfach-instanziierter Teilprozess ins Leben gerufen, dessen Aufgabe es ist, für die jeweiligen Empfangssysteme die passenden Schnittstellen zu ermitteln, die Daten entsprechend der Zielstrukturen anzupassen und sie schließlich zu versenden. Es ist genau an dieser Stelle, an der die Spezialfähigkeiten des ESB's genutzt werden.

Die Empfangssysteme wiederum verarbeiten die übermittelten Daten und senden ihrerseits eine Antwort an den ESB zurück. Sowohl durch die Schnittstelleninforma-

tionen der empfangenen Nachricht als auch durch die Absenderdaten kann der ESB den systemzentrischen Prozess als Empfänger ermitteln. Vor der Übergabe der Daten an den wartenden Prozess erfolgt durch einen *Mapping*-Schritt allerdings noch die Umsetzung in das kanonische Datenmodell. Hierfür wurde die Zielschnittstelle zuvor in der *Schnittstellen-Ermittlung*-Aufgabe bestimmt. Der systemzentrische Prozess aktualisiert für jede empfangene Nachricht die Cross-Referenz-Tabelle. Sind sämtliche Antwortnachrichten auf diese Weise über den ESB beim systemzentrischen Prozess eingetroffen, kann dieser schließlich die Antwortnachricht erstellen und an den Prozess der Verbundanwendung weiterreichen. Der Zyklus ist damit abgeschlossen.

Dem Modell ist zu entnehmen, dass die jeweiligen Aufgaben nun wieder klar aufgeteilt sind, also erneut eine Komponentisierung stattgefunden hat, um auf diese Weise eine weitere Flexibilitätssteigerung zu erzielen. Der Einsatz eines ESB's im Kontext von Verbundanwendungen ist zu empfehlen, da zum einen, wie gezeigt, die Gesamtarchitektur abgerundet werden kann und sich zum anderen die Implementierungszeiten verkürzen, da aufwändige Anbindungen an Drittsysteme (technische Anbindung, Protokolle, Datenformate) dem ESB und seinen Werkzeugen überlassen bleiben, die dafür konzipiert und optimiert wurden. So muss der systemzentrische Prozess bei Änderungen in der Systemlandschaft nun nicht mehr angepasst werden, da die gesamte Kommunikation mit den Backend-Systemen und deren proprietären Schnittstellen über den ESB abgewickelt wird. Da sich Änderungen in der System-Konstellation zwangsläufig in den Routing-Tabellen des ESB's niederschlagen müssen, profitiert der systemzentrische Prozess automatisch von den Anpassungen, da dieser bei der Empfänger-Ermittlung genau auf diese vom ESB bereitgestellten Informationen zurückgreift. Auch die Verwendung der Datentypen verläuft recht harmonisch: sowohl der Prozess der Verbundanwendung als auch der systemzentrische Prozess arbeiten ausschließlich auf dem kanonischen Datenmodell. Erst bei der Kommunikation mit den Backend-Systemen müssen Transformationen vorgenommen werden, die nun aber im ESB gekapselt sind und dadurch lokal leicht änderbar verwaltet und angepasst werden können.

Im weiteren Verlauf dieser Arbeit steht zunächst in dem nun folgenden Abschnitt eine Diskussion über den Einsatz von Repositories im Zusammenhang mit Verbun-

danwendungen im Mittelpunkt der Betrachtungen. Abgeschlossen wird das Kapitel Architektur von Verbundanwendungen schließlich mit einer Auseinandersetzung über das Thema, wie die in einer Verbundanwendung verwendeten Services idealerweise verwaltet werden können. In Kapitel 4 wird dann durch eine konkrete Implementierung als Proof-of-Concept (POC) die Umsetzbarkeit der vorgeschlagenen Grundarchitektur demonstriert.

## 3.4 Service Repositories und Verbundanwendungen

Obwohl bereits in der Einleitung darauf hingewiesen wurde, dass typische SOA-Aspekte, die nicht unmittelbar für die Architektur einer Verbundanwendung von Relevanz sind, auch in dieser Arbeit nicht weiter vertiefend behandelt werden sollen, muss dennoch auf ein fundamentales Unterscheidungsmerkmal bei der Verwendung von Service Repositories im Zusammenhang mit Verbundanwendungen eingegangen werden. Dabei sind Service Repositories wiederum von Service Registries zu unterscheiden. Aus Josuttis 2008, S. 287 stammen die nun folgenden Definitionen zur besseren Unterscheidung der beiden zentralen Service-Verwaltungsdatenbanken:

- Ein Repository verwaltet Services aus *fachlicher Sicht* (Josuttis 2008, S. 287). Dabei unterstützt ein Repository die Verwaltung von Schnittstellen, Verträgen, Service Level Agreements (SLAs), Abhängigkeiten usw., um Services zu identifizieren, zu entwerfen und zu implementieren (S. 287). In einem Repository sollen alle Aspekte eines Service abgelegt werden, die aus fachlicher Sicht relevant sind (S. 287). Die Informationen sind dabei unabhängig von der verwendeten Infrastruktur und den Implementierungsdetails (S. 287).

  Ein typisches Beispiel für ein Repository, das dieser Definition folgt, ist das von der SAP entworfene Enterprise Services Repository (ESR). In ihm sind beispielsweise die mehr als 2.800 Enterprise Services der SAP-Standardanwendungen aufgeführt, aus denen sich SAP-Kunden bei der Erstellung von Verbundanwendungen bedienen können. Aber auch Nicht-SAP-Kunden können sich anhand der Enterprise Services orientieren, welche betriebswirtschaftlichen Standardkomponenten bereits existieren und die folglich nicht nochmal neu ausprogrammiert werden müssen. Weitere Details zum

ESR sind unter SAP 2010b nachzulesen.

- Eine Registry verwaltet Services aus *technischer Sicht*, umfasst also alle technischen Aspekte, die zum Betrieb der Services notwendig sind (Josuttis 2008, S. 287). Dazu gehören insbesondere Laufzeitinformationen wie zum Beispiel die Adressen der Services, unter denen sie zu erreichen sind (S. 287). Zur Laufzeit können diese Informationen gelesen und Aufrufe dynamisch um- bzw. weitergeleitet werden (S. 287). Klassische Beispiele für Registries sind sämtliche Implementierung des UDDI-Standards (Universal Description, Discovery, and Integration).

Für die weitere Diskussion im Zusammenhang mit Verbundanwendungen soll insbesondere die Rolle des Repositories genauer betrachtet werden. Interessanterweise wird in der SOA-Literatur stets auf die Bedeutung des Repositories bei der *Service-Suche* hingewiesen: welche Services existieren bereits, welche fachlichen Funktionalitäten werden zur Verfügung gestellt und wie sehen die dazugehörigen Schnittstellen aus? Dies sind typische Fragestellungen, die ein Repository beantworten kann.

Für Verbundanwendungen allerdings muss das Repository eine gänzlich neue Rolle übernehmen: nämlich die der *Bereitstellung der Servicevertrag-Informationen* der jeweiligen Verbundanwendungen. Hierbei handelt es sich ebenfalls um fachliche Schnittstelleninformationen, allerdings werden diese Informationen seitens des Servicevertrag-Programmierers herangezogen um zu wissen, welche Funktionalität er konkret zu implementieren hat, damit die Composite Application korrekt ablaufen kann. Dazu benötigt er genau die betriebswirtschaftlichen Informationen, von denen in der Definition des Repositories die Rede war. Das Service Repository ist von daher also genau das richtige Register für diese Daten, da die Serviceverträge fachlich motiviert sind.

Die Verwendung des Service Repositories wird auf diese Weise erweitert: neben der klassischen Nutzung zum Auffinden von Services jetzt also auch die Verwendung zur Außendarstellung von Verbundanwendungen. Letztendlich wird durch das Repository die Frage beantwortet, von welchen Services eine Composite erwartet, dass sie in der

Servicevertrag-Implementierungsschicht umgesetzt werden und welche Fachlichkeit dadurch zu erbringen ist. Die Verträge können dabei, je nach Version der Verbundanwendung, unterschiedlich ausfallen. Von daher ist ein Versionsmanagement für Serviceverträge innerhalb des Repositories unumgänglich. Heutige Repositories können mit einer solchen Funktionalität aufwarten, so dass unterschiedliche Versionen kein allzu großes Problem sind. Bei der Einführung einer Verbundanwendung in einer bestimmten Version bekommt der Entwickler der Servicevertrag-Implementierungsschicht für eine bestimmte Verbundanwendungsversion automatisch alle umzusetzenden Serviceverträge aufgelistet. Die Liste dient dem Entwickler dann zur Orientierung während der Implementierungsphase.

In diesem Zusammenhang muss neben den bereits erwähnten Unterschieden zwischen SOA-Anwendungen und Verbundanwendungen gerade im Bereich der Services auch auf deren Gemeinsamkeiten eingegangen werden. So setzen beide Anwendungsarten folgende Eigenschaften von Services (Josuttis 2008, S. 38ff) voraus (Services werden in diesem Zusammenhang verstanden als Dienste, die von den jeweiligen Anwendungen extern aufgerufen werden (Konsumentensicht) und nicht die Dienste, die seitens der Applikationen zur Verfügung gestellt werden):

- Abgeschlossenheit: die in einer Verbundanwendung verwendeten Services müssen in sich abgeschlossen und damit unabhängig von anderen Services und dem allgemeinen Systemzustand sein (Josuttis 2008, S. 39).
- Grobgranularität: der Serviceumfang umfasst durchaus komplexe betriebswirtschaftliche Abläufe und geht damit deutlich über einfache Datenbank-Operationen hinaus (Josuttis 2008, S. 39). Welchen Umfang Services gerade im Unternehmenseinsatz annehmen können, zeigen die von SAP veröffentlichten Enterprise Services im Enterprise Services Repository (SAP 2010b).
- Sichtbarkeit/Auffindbarkeit: Wiederverwendbarkeit von Services ist nur dann gewährleistet, wenn deren Vorhandensein auch bekannt ist (Josuttis 2008, S. 41). Von daher spielen die bereits erwähnten Service Repositories und Registries eine wichtige Rolle bei der Auffindbarkeit von Services.
- Zustandslosigkeit: von Zustandslosigkeit ist die Rede, wenn zwischen zwei aufeinanderfolgenden Aufrufen innerhalb der Service-Implementierung kein

Zustand in Form temporärer lokaler Variablen gehalten werden muss (Josuttis 2008, S. 41). Sie können nach dem Service-Aufruf gelöscht werden. Diese Art der Zustandslosigkeit ist von dem Zustand zu unterscheiden, der sich nach einem Serviceaufruf beispielsweise in einer Datenbank manifestiert hat. So ist der Service zur Anlage eines Auftrags in sich zustandslos, da die temporären Variablen nach dem Aufruf gelöscht werden können. Allerdings wird in der Datenbank ein entsprechendes Auftragsobjekt hinterlegt, auf das ein Auftragsänderungsservice, der selbst auch wieder zustandslos ist, zugreifen und verändern kann. Diese Art von Zustand wird es auch von an sich zustandslosen Services immer geben.

- Idempotenz: Idempotenz ist gewährleistet, wenn ein Service mit denselben Daten mehrfach aufgerufen werden kann, die eigentliche Operation aber nur einmalig ausgeführt wird (Josuttis 2008, S. 41). Gerade in Web-Anwendungen ist Idempotenz von großer Bedeutung: Endanwender betätigen bei einer Bestellung beispielsweise des Öfteren den *Sende*-Knopf zum Versand der Bestelldaten, wenn nicht sofort eine Rückmeldung am Bildschirm erscheint, dass die Daten erfolgreich verschickt wurden. Ohne Idempotenz wäre eine Mehrfachbestellung des aktuellen Warenkorbs die Folge. Mit Idempotenz können derartige Situationen erkannt und behandelt werden. Im Rahmen von Kapitel 5 (Ergänzende Konzepte zur Unterstützung der Architektur von Verbundanwendungen) wird auf diese Eigenschaft noch detaillierter eingegangen.
- Wiederverwendbarkeit (Josuttis 2008, S. 42): dies ist wohl eines der wichtigsten Ziele sowohl von SOA- als auch von Verbundanwendungen – die Wiederverwendbarkeit existierender Funktionalitäten. Von daher ist allein aus Gründen zur Vermeidung von Redundanzen diese Eigenschaft erstrebenswert.

Neben diesen Gemeinsamkeiten gibt es auch kleinere Unterschiede: so legen Verbundanwendungen keinen Wert auf die Implementierung eines Dienstes als Web Service. Viel wichtiger ist die abgeschlossene fachliche Funktionalität des Service. Wie dieser technologisch zu erreichen ist oder wie die Schnittstelle dargestellt wird, ist unerheblich. Moderne Infrastrukturprodukte wie ESBs erlauben über entsprechende Adapter ohnehin Protokolltransformationen, so dass auch Systeme in Verbundanwendungen eingebunden werden können, die technologisch noch nicht auf Web Services

ausgerichtet sind. Dasselbe gilt für die Schnittstellen selbst, sofern sie noch nicht im WSDL-Format vorliegen. Auch hier kann durch entsprechende Produkte eine Transformation vorgenommen werden. So ist es in der SAP-Produktfamilie möglich, sämtliche fachlichen Funktionalitäten, die über die SAP-proprietären Protokolle und Schnittstellen RFM (Remote Function Module) und BAPI (Business Application Programming Interface) zur Verfügung gestellt werden, mittels eines Proxy-Generators in WSDL-Beschreibungen zu transformieren und über das SOAP-Protokoll zugänglich zu machen. Der Wiederverwendungsgrad existierender Funktionalität erhöht sich dadurch doch recht deutlich im Vergleich zu reinen SOAP/WSDL-basierten Diensten.

Ein weiterer nicht ganz unwichtiger Unterschied ist die Bedeutung von Governance-Prozessen für Verbundanwendungen. Während für SOA im Allgemeinen die Frage nach der Wiederverwendbarkeit von Services und damit verbunden das Etablieren von Prozesse innerhalb der Unternehmen zur Bestimmung der richtigen Service-Granularität, das Aussehen der Service-Schnittstellen und der dabei verwendeten Datentypen, die Erstellung von Guidelines zur Service-Erstellung, die Überwachung der Service-Verwendung und die Betreuung des Dienstes über seinen gesamten Lebenszyklus hinweg einen sehr zentralen Aspekt darstellen, ist dies für Verbundanwendungen eher von untergeordneter Bedeutung. In dieser Hinsicht sind Composite Applications in erster Linie Service-Konsumenten. Die während der Entwicklung der Composite erstellten internen Services dienen primär der Umsetzung der neuen fachlichen Anforderungen. Eine Wiederverwendung steht erst einmal nicht im Vordergrund. Sollte eine Wiederverwendung zu einem späteren Zeitpunkt in Frage kommen, so können die im Unternehmen etablierten Governance-Prozesse auch dann noch angewandt und die dadurch notwendig werdenden Änderungen über Refactoring in die Verbundanwendung eingebracht werden. Hintergrund dieses Vorgehens ist die außerordentliche Bedeutung von Composites für die Unternehmen und damit verbunden das Ziel ihrer schnellstmöglichen Implementierung. Per Definition sollen Composite Applications ja gerade die innovativen Prozesse unterstützen, die dem Unternehmen Wettbewerbsvorteile gegenüber der Konkurrenz bringen. Je eher die Lösung daher produktiviert werden kann, umso besser ist dies für das Unternehmen. Von daher sollte der Implementierungsprozess so schlank wie möglich gehalten werden, was im Widerspruch zu den eher langwierigen SOA-Governance-Prozessen steht.

## 3.5 Service-Management: unterschiedliche Ansätze im Vergleich

Im vorangegangenen Abschnitt zeigte die Diskussion über die Unterschiede zwischen einem Service-Repository und einer Services-Registry, wo welche Informationen über Services abgelegt und wie diese entweder während des Designs serviceorientierter Software oder zur Laufzeit geeignet eingesetzt werden können. Die Ausführungen berücksichtigten dabei die momentan im praktischen Einsatz befindlichen Lösungen. In der Forschung wird derzeit allerdings an weiterführenden Konzepten gearbeitet, um die Verwaltung, die Suche und den Aufruf von Services weiter zu optimieren und zu dynamisieren. In diesem Abschnitt soll daher kurz ein Blick auf die laufenden Projekte geworfen und damit ein Ausblick auf zu erwartende Verbesserungen in diesem Bereich gegeben werden.

Der erste Ansatz beruht auf der sogenannten Drei-Schema-Architektur für das Service-Management (Fischer, Link, Ortner & Zeise 2010). In dem Papier wird eine Organisationsstruktur für Services vorgeschlagen, die sich an der entsprechenden Schema-Architektur von Datenbanksystemen orientiert. Konkret werden drei Schemata diskutiert, die in Schichten aufeinander aufbauen und die wie in Abbildung 14 dargestellt miteinander interagieren.

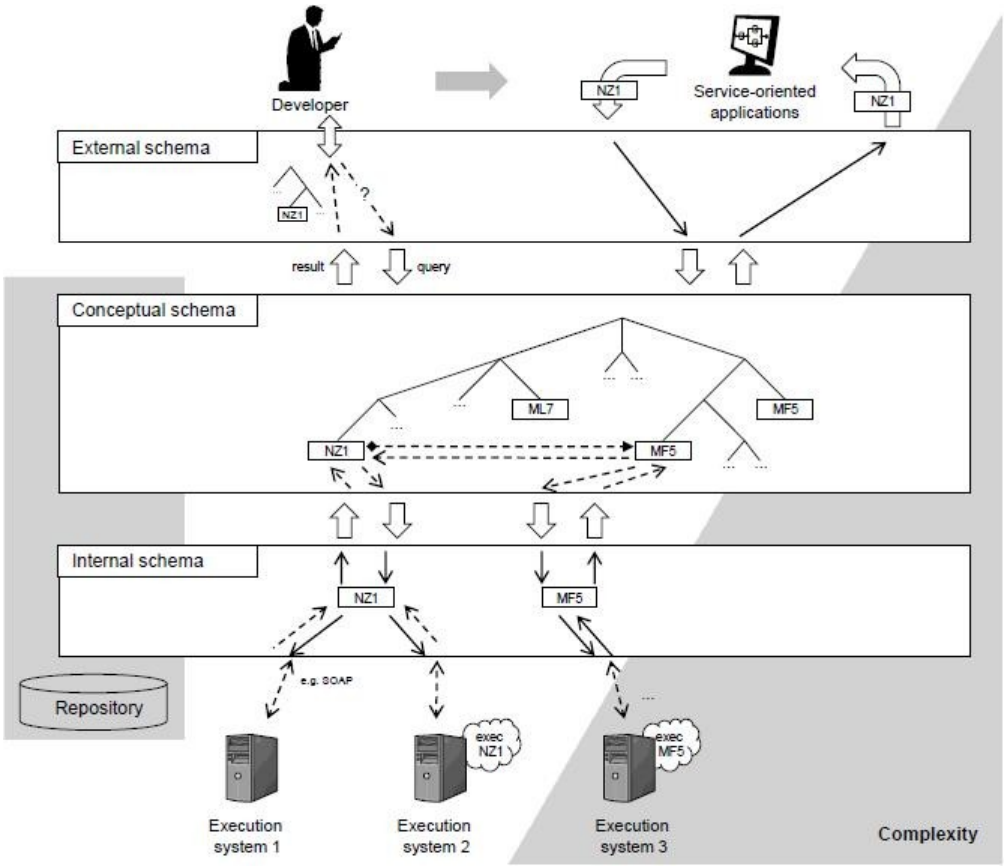



***Abbildung 14: Drei-Schema-Architektur für das Service-Management (aus Fischer, Link, Ortner & Zeise 2010)***

Jede dieser Schichten ist in sich abgeschlossen und bietet über wohldefinierte Schnittstellen Dienste an. Durch die Schichtung werden drei wesentliche Ziele verfolgt:

1. Konsistenter Zugriff auf die Services, repräsentiert durch das externe Schema, um die Nutzung von Services zu vereinfachen.
2. Konsistente Beschreibung von Services, unterstützt durch das konzeptionelle Schema, das die Suche nach Services mit bestimmten Eigenschaften gemäß der Anforderungen der zu erstellenden Applikation erleichtert.
3. Unabhängigkeit von konkreten Service-Implementierungen und damit die Freiheit, Services zu optimieren oder auszutauschen, ohne Anpassungen an konsumierenden Anwendungen vornehmen zu müssen. Dies wird durch das interne Schema erreicht, das letztendlich die physischen Service-Instanzen verwaltet.

Die drei Schemata bilden die Grundpfeiler des sogenannten Servicebase Management-Systems (in Anlehnung an Database Management Systems). Dieses wird hauptsächlich von zwei Arten von Konsumenten genutzt. Zum einen vom Anwendungsentwickler, der zur Umsetzung der Applikationsanforderungen passende Services finden muss und hauptsächlich suchend auf das System zugreift. Als Ergebnis dieser Suche erhält er eine konsistente Schnittstelle (das externe Schema), über die er den Service aufrufen kann. Zum anderen die aufrufende Applikation selbst, die zur Laufzeit das System für den Zugriff auf die Services benötigt.

Offensichtlich übernimmt bei diesen Interaktionen das konzeptionelle Schema eine besondere Rolle. Es umfasst nicht nur eine kategorisierte Übersicht aller Services mit ihren Metainformationen, sondern entkoppelt darüber hinaus die Nutzung der Services von ihren Implementierungen, indem es die Vermittlung zwischen dem externen Schema, über den der Aufruf erfolgt, und der konkreten Implementierung repräsentiert durch das interne Schema übernimmt.

Damit das konzeptionelle Schema diesen Aufgaben gerecht werden kann, muss es zwangsläufig einige Anforderungen erfüllen bzw. bestimmte Inhalte bereitstellen. Fischer, Link, Ortner & Zeise führen in ihrem Papier u.a. die folgenden Punkte auf:

- Konsistente semantische Beschreibung der Services (u.a. die Beantwortung der Frage, welche Aufgabe durch den Service erfüllt wird)
- Verknüpfung zwischen externem und internem Schema
- Verbindungen zu anderen Services (welche Services werden während der Ausführung vom aktuellen Service genutzt? Welche anderen Services verwenden den aktuellen Service?). Dieser Punkt verdeutlicht zudem, dass das Servicebase Management System nicht nur atomare Services verwaltet, sondern auch Services unterstützt, die sich selbst wieder weiterer Services bedienen (Composite Services)
- Voraussetzung für die Nutzung des Services
- Nachbedingungen nach Beendigung des Aufrufs
- Seiteneffekte bei der Nutzung des Service (z.B. Datenbankänderungen)
- Kosten der Service-Nutzung
- Erwartete Antwortzeiten

Dabei spielen beim Sammeln derartiger Informationen Redundanzen keine Rolle, da Services häufig nicht eindeutig kategorisiert werden können. Im Gegenteil: Redundanzen helfen dem Entwickler vielmehr bei seiner Suche und sind daher sogar erwünscht.

Grundvoraussetzung für das Funktionieren dieses Ansatzes ist die Bereitstellung obiger Metadaten in geeigneter, maschinen-auswertbarer Form, die anschließend über eine SQL-ähnliche Sprache durchsucht und verändert werden können. Derzeit wird an einer konkreten technischen Umsetzung der drei Schemata als auch an der Implementierung eines ersten Prototypen gearbeitet.

Das Problem der Bereitstellung von Metadaten und semantischen Informationen zu Services in maschinen-auswertbarer Form ist auch Gegenstand anderer Forschungsprojekte. Insbesondere die Untersuchungen rund um semantische Web-Technologien haben in den letzten Jahren große Fortschritte gemacht. Die Arbeiten von Stein 2009 und Lautenbacher 2009 gehen dabei sehr detailliert auf die damit verbundenen Standards und Lösungsansätze ein. Im Wesentlichen geht es um den automatisierten Abgleich von betriebswirtschaftlichen Anforderungen, so wie sie die Fachabteilung formuliert, gegenüber der Menge der verfügbaren Services, die diese Anforderungen zu implementieren haben. Dazu müssen sowohl die Anforderungen seitens der Fachexperten genauso maschinenlesbar zur Verfügung stehen wie die Metainformationen über die Services. Die Art und Weise, wie Anforderungen und Metadaten darzustellen sind, wird formal in sogenannten Ontologien festgelegt. Dabei können sich die Ontologien der Fachanforderungen und die der Service-Metadaten durchaus unterscheiden. Sie werden ja auch von unterschiedlichen Personengruppen zur Verfügung gestellt (fachliche Anforderungen von den Fachexperten, Service-Metadaten von den Experten der IT-Abteilung). Damit das Ziel einer automatisierten Zuordnung von Anforderung zu einem Service oder einer Kette von Services ermöglicht wird, müssen die unterschiedlichen Ontologien aufeinander abgebildet werden. Erst auf Basis dieser Informationen können Rechner dann korrekte Schlussfolgerungen auf benötigte Services zur Implementierung spezifischer Anforderungen ziehen. Dieser Vorgang der automatisierten Zuordnung von fachlichen Anforderungen zu einem oder einer Reihe von Services wird auch mit *Machine Reasoning* bezeichnet.

Auf Basis dieser grundlegenden Idee haben sich Komplettansätze bis hin zu einem semantischen Business Process Management entwickelt. Dabei werden die Prozessmodelle wie gewohnt in BPMN modelliert. Allerdings ergänzt der Modellierer sein Modell um semantische Anmerkungen, die seine fachlichen Anforderungen an einen Prozessschritt oder ein Ereignis repräsentieren. Hierzu verwendet er die bereits erwähnten Ontologien, wovon es wiederum eine Vielzahl unterschiedlicher Ausprägungen geben kann. Beispiele sind Ontologien für Prozesse, Unternehmens-Organisation, verschiedenste Fachdomänen oder eben Web Services. Schon während der Modellierung erfährt der Fachexperte eine Unterstützung vom System, indem durch eine semantische Suche nach existierenden Prozessen, Prozessfragmenten oder Services in einem semantischen Repository eine automatisierte Vervollständigung seines Modell ermöglicht wird. Auch eine Verifikation des Prozesses nach semantischen Kriterien wird unterstützt.

Ein besonderes Feature stellt die automatisierte Komposition von Web Services zur Erfüllung einer betriebswirtschaftlichen Funktionalität dar. Kann bei der Suche im Repository zur Umsetzung einer Anforderung kein geeigneter Service gefunden werden, der die Funktionalität vollständig abdeckt, so versucht das System durch Kombination verschiedenster Services die geforderte Leistung zu erbringen. Dies ist möglich, da auch die Ein- und Ausgaben der Services ontologisch annotiert sind und in den Algorithmen des *Machine Reasoning* verwendet werden können.

So vielversprechend diese Ansätze auch klingen mögen, so dürfen auch die Probleme nicht verschwiegen werden. Lautenbacher 2009 führt gleich eine Vielzahl von Herausforderungen auf, deren sich die Forschung stellen muss. Dabei ist auffallend, dass in den meisten Fällen die Ontologien selbst im Mittelpunkt stehen (Lautenbacher 2009, S. 43ff):

1. Ontologien sind äußerst fragil und ändern sich laufend aufgrund der domänenspezifischen Dynamik (neue Elemente kommen hinzu während andere verschwinden).
2. Unternehmen sehen keinen wesentlichen Vorteil in der aufwändigen und kostspieligen Entwicklung neuer Ontologien, wenn sie sich davon keinen entsprechenden Vorteil versprechen.

3. Ontologien werden in der Regel von einer kleinen Gruppe von Experten erstellt, doch die Konsumenten stellen sich zurecht die Frage, inwiefern sie der Ontologie trauen können oder welche Freiheiten sie bei ihrer Erweiterung haben.
4. Die Einarbeitung in neue Ontologien gestaltet sich schwierig, da der Nutzer bei der Ausgestaltung selbst nicht mitgewirkt hat und sich nun ausschließlich anhand formaler Vorgaben orientieren muss.
5. Wie steht es um den Schutz des geistigen Eigentums des Ontologie-Erstellers bzw. des Ersteller-Gremiums?
6. Das Problem der geeigneten Ontologie-Größe: wie groß muss eine Ontologie sein, damit sie einen deutlichen Mehrwert liefert und wie groß darf sie maximal sein, damit sie nicht unhandlich und somit ungeeignet für den Einsatz wird?
7. Wer fühlt sich berufen, Ontologien zu entwerfen?

Von daher ist es nur zu verständlich, dass semantische Webtechnologien in der Praxis noch mit gewissen Vorbehalten betrachtet werden, obwohl es in der Forschung bereits vielversprechende Ansätze gibt. Ein Grundproblem ist sicherlich die *zentrale* Entwicklung der Ontologien. Immer dann, wenn Gremien zentral an Vorgaben für Spezifikationen, Formate oder Ähnliches arbeiten, besteht stets die Gefahr, dass aufgrund unterschiedlichster Meinungen, Bedürfnisse und Sichtweisen entweder überhaupt keine Ergebnisse erzielt oder nur unbefriedigende Kompromisslösungen abgeliefert werden. Von daher ist zu überlegen, ob ein dezentraler Ansatz nicht zu bevorzugen ist.

Dazu beschreiben Vanderhaeghen, Fettke & Loos 2010 in ihrem Papier Ansätze, wie sich die dezentralen Gestaltungsprinzipien des Web 2.0 nutzbringend einsetzen lassen. Dabei spielen die Aspekte *Selbstorganisation* und *kollektive Intelligenz* eine besondere Rolle (Vanderhaeghen, Fettke & Loos 2010, S. 19). Diese beiden Eigenschaften haben sich in Web 2.0-Anwendungen wie Wikipedia, den vielen Bewertungsplattformen für Produkte, Hotels, Reisen etc. oder bei den vielfach verlinkten Blogs längst bewährt. Auch wenn es in dem Papier um die Anwendung dieser Charakteristika im Kontext des Prozessmanagements geht, so lassen sich diese Ideen auch

für die semantische Beschreibung von Services und der intelligenten Suche nach geeigneten Services für eine bestimmte betriebswirtschaftliche Anforderung umsetzen. Die Kernidee ist dabei die des dezentralen Taggings von Services. Metadaten werden in Form von Tags hinzugefügt, so wie dies beispielsweise für Bookmarks bei der Internetplattform Delicious erfolgt. Die initiale Versorgung mit Tags erfolgt vom Verantwortlichen des Service selbst. Er kennt die Funktionalität am besten und kann grundlegende Informationen über den Service beisteuern. Danach steht es jedem Nutzer frei, ergänzende Tags hinzuzufügen und diese mit anderen zu teilen. Nutzungsvoraussetzung, Nachbedingungen, Erfahrungen bei der Nutzung, Fehler, Laufzeitverhalten, verschiedenen Sprachen, Bewertungen, Nutzungshäufigkeit, Version, all dies kann in Form von Tags hinzugefügt werden. Besonders interessant ist die Verlinkung von Services (und damit der Tags) um zu veranschaulichen, wie durch Kombination von Services höherwertige Dienste erbracht werden können. Auch dieser neu entstandene höherwertige Service wird analog zu den atomaren Services mit Tags versehen. Auf diese Weise wird das kollektive Wissen zum Nutzen aller zur Verfügung gestellt. Zudem hat jeder Nutzer die Möglichkeit, domänenspezifische Begriffe nach seinen Bedürfnissen anzupassen. So ist ein Kunde eines Rechtsanwalts ein Klient und ein Service, der zuvor nur bei Verwendung des Suchbegriffs *Kunde* gefunden werden konnte, ist nach der Ergänzung auch unter *Klient* zu finden. Da keine formalen Vorschriften einzuhalten sind, lassen sich solche Ergänzungen über eine Webplattform von jedermann leicht einbringen. Schließlich erlaubt eine google-ähnliche Suche das rasche Auffinden geeigneter Services. Über die Zeit entwickelt sich somit ein hochintelligentes Netzwerk von semantischen Service-Beschreibungen, das aufgrund seines dezentralen Charakters umfassender und leistungsfähiger ist, als dies über ein zentral gesteuertes Gremium jemals erreicht werden kann.

In diesem Abschnitt wurden einige Ansätze für das Service-Management diskutiert. Ob drei-Schema-Architektur, semantische Web-Technologien oder Selbstorganisation mit kollektiver Intelligenz: man darf gespannt sein, wie sich die Verwaltung von Services in Zukunft weiterentwickeln wird.

# 4 Implementierung der Grundarchitektur von Verbundanwendungen

Zur Verdeutlichung der im vorangegangenen Kapitel dargestellten Konzepte zur Grundarchitektur von Verbundanwendungen, wird im Folgenden konkret eine Composite Application entwickelt, die die wesentlichen Ideen adressiert und deren Umsetzung veranschaulicht. Dabei wird insbesondere auf die Implementierung der losen Kopplung unter Verwendung der BPMN sowohl für die Verbundanwendung selbst als auch für die Servicevertrag-Implementierungsschicht Wert gelegt. Auf diese Weise wird die Durchführbarkeit der beschriebenen Aspekte unter Beweis gestellt.

Aufgrund der bisher geführten Diskussionen ist deutlich geworden, dass eine vollständige Verbundanwendung Prozesse, Benutzeroberflächen, Objekte (einschließlich ihrer Persistenz) und Services umfasst. Diese Bestandteile gilt es nun auch für die Beispielanwendung zu implementieren. Darüber hinaus muss die Interaktion zwischen Verbundanwendung und der Servicevertrag-Implementierungsschicht über entsprechende Schnittstellen in Form von Verträgen ausgedrückt und eine passende Umsetzung dieser Verträge realisiert werden. Bevor auf die eigentliche Entwicklung der Anwendung eingegangen wird, folgt zunächst ein weiterer Abschnitt zur Business Process Model and Notation (BPMN), der insbesondere auf die besondere Bedeutung der BPMN für die Implementierung von Prozessen innerhalb von Verbundanwendungen eingeht.

## 4.1 Besondere Bedeutung der BPMN für die Implementierung von Verbundanwendungen

Bereits in Kapitel 2.1 (Die Rolle der BPMN (Business Process Model and Notation) für Verbundanwendungen - Grundlagen) wurde auf die Grundlagen der BPMN zur Modellierung von Verbundanwendungen hingewiesen. Der Fokus lag dabei auf den Elementen der BPMN, die der fachlichen Modellierung dienten. In diesem Abschnitt werden nun die weitergehenden Konzepte von BPMN erläutert, die insbesondere beim Einsatz in der Servicevertrag-Implementierungsschicht eine Rolle spielen. Hier kommen die technischen Neuerungen der Version 2.0 zum Tragen, die sich mit der Ausführbarkeit auseinandersetzen und die jegliche Transformationen in andere ausführbare Sprachen wie BPEL überflüssig werden lassen (zu Transformationen in andere Sprachen vgl. auch Lautenbacher 2009 und Stein 2009). In dieser Arbeit wird

sion ausdrucksstark genug ist, um auch technische Abläufe präzise zu modellieren und dadurch die Grundlagen geschaffen wurden, BPMN-Modelle zur Ausführung zu bringen. BPMN wird somit zu einer grafischen Programmiersprache. Unter diesem Aspekt ist die nun folgende Diskussion zu verstehen, deren Schwerpunkt dabei auf den folgenden Konzepten der BPMN liegt:

- Ereignisse
- Parallelverarbeitung
- Ausnahmebehandlung
- Transaktionen und Kompensation
- Kollaborationen und Nachrichtenaustausch

### *4.1.1 Ereignisse*

Eine besondere Stärke von BPMN liegt in ihrer Ereignisorientierung. Wie bereits in Abschnitt 2.1.1.2 (Ereignisse) erwähnt, sind nicht weniger als 13 verschiedene Ereignistypen unterscheidbar. Erwähnt wurden bereits das Nachrichtenereignis (  ), das beim Eintreffen von Nachrichten aktiviert wird, und das Zeitereignis (  ). Weitere wichtige Ereignistypen im Zusammenhang mit Verbundanwendungen sind:

- Die Eskalation (  ) zur Signalisierung von betriebswirtschaftlichen Ausnahmesituationen
- Das Fehlerereignis (  ) zur Anzeige technischer Probleme
- Das Abbruchereignis (  ) im Zusammenspiel mit der Kompensation () zur Abwicklung von Transaktionen
- Das Signal () zur gleichzeitigen Kommunikation innerhalb und außerhalb des Prozesses über Publish-Subscribe-Mechanismen
- Die Terminierung () zur sofortigen Prozessbeendigung unabhängig davon, ob noch Token im Prozess vorhanden sind oder nicht.

Neben den Ereignistypen können noch folgende Merkmale unterschieden werden:

- Positionierung der Ereignisse im Prozess: Ereignisse können an unterschiedlichen Stellen des Prozessflusses eingebracht werden. Dabei ist für die unter-

schiedlichsten Ereignistypen nicht jede Position geeignet. Ereignisse am Anfang des Prozesses werden als Startereignis bezeichnet, am Ende des Prozesses als Endereignis, alle anderen als Zwischenereignis.

- Verwendung im Prozessfluss: direkt im normalen Sequenzfluss eingebettet oder an Aktivitäten angeheftet
- Unterbrechend/nicht-unterbrechend: diese Unterscheidung ist für angeheftete Ereignisse von Bedeutung und lässt den Leser des Modells erkennen, ob bei Eintritt des Ereignisses die dazugehörige Aktivität unterbrochen wird oder nicht
- Auslösend oder Eintretend, ob also das Ereignis vom Prozess selbst gesendet oder empfangen wird

Durch die Verwendung von Ereignissen wird es dem Modellierer also ermöglicht, auf außergewöhnliche Situationen zu reagieren bzw. außergewöhnliche Situationen selbst anzuzeigen. Er muss sich aktiv mit der Frage auseinandersetzen, wie der Prozess zu handeln hat, wenn bestimmte Ereignisse während des Prozessablaufs eintreten. Genauso kann er das Auftreten von Ausnahmen seiner Umgebung durch Ereignisse signalisieren. In der Regel wird es sich dabei stets um Prozesssequenzen handeln, die den *Happy Path* verlassen. Wie noch zu sehen sein wird, ermöglicht BPMN dabei eine sehr kompakte Darstellung der Ereignisbehandlung und motiviert den Modellierer, auch die Fehlerfällen zu betrachten. Gerade im Hinblick auf die Ausführbarkeit von Prozessmodellen bedeutet diese Art der Modellierung eine enorme Zeitersparnis, wird doch die Programmierung von Ausnahmebehandlungen seitens der Entwickler eher als lästig, mühsam und unangenehm empfunden. Durch eine derartige Modellierung wird die Ausnahmebehandlung hingegen zu einem normalen Bestandteil des Prozesses. Die eigentliche Umsetzung und Überwachung der Ereignisbehandlung wird dann von den Frameworks übernommen, die die BPMN implementieren.

In den nachfolgenden Abschnitten werden Events weiterhin intensiv genutzt. In diesem Zusammenhang wird dann auch detailliert auf weitere Unterschiede eingegangen, sowie die Verwendung und Nutzen der Ereignisse in dem jeweiligen Umfeld geschildert.

### 4.1.2 Parallelverarbeitung

Die Parallelverarbeitung ist für Verbundanwendungen von besonderer Bedeutung, da durch das Aktivieren paralleler Prozesspfade die Prozessdurchlaufzeiten optimiert und die Rechnerressourcen besser ausgelastet werden. Seitens der BPMN wird dies auf vielfältige Weise unterstützt. Von den bereits diskutierten BPMN-Elementen sind in diesem Zusammenhang das parallele Gateway und das inklusiven Gateway besonders hervorzuheben.

Durch das parallele Gateway werden unabhängig von Bedingungen Pfade aktiviert und somit eine maximale Parallelisierung erreicht. In Abbildung 15 beispielsweise werden drei Systeme gleichzeitig über entsprechende Nachrichten mit Daten versorgt.

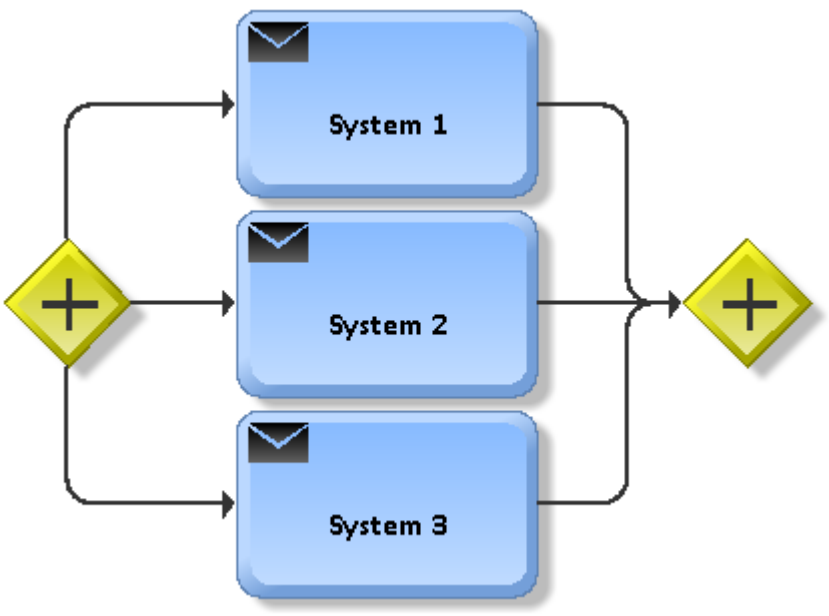



***Abbildung 15: Paralleles Gateway***

Gerade die Servicevertrag-Implementierungsschicht profitiert von der Parallelisierung, da die parallelen Sequenzflüsse unabhängig voneinander abgearbeitet und dadurch optimale Reaktionszeiten erreicht werden. Müssen z.B. zur Erfüllung der betriebswirtschaftlichen Funktionalität *Kundendaten lesen* mehrere Systeme aufgerufen werden, so kann dies gleichzeitig erfolgen und nach Eintreffen aller Antworten die zusammengetragenen Informationen geschlossen an den Konsumenten der Funktion zurückgeliefert werden.

Das inklusive Gateway ermöglicht ebenfalls eine Parallelverarbeitung. Allerdings sind bei ihm die ausgehenden Sequenzflüsse stets an Bedingungen gebunden. Dennoch erlaubt das inklusive Gateway die elegante Modellierung einer typischen Prozesssituation: neben den bedingten Ausgangsflüssen kann ein Pfad auch ohne Bedingung angegeben werden. Dieser wird folglich immer ausgeführt, zusätzlich zu denen, deren Bedingungen erfüllt sind. Die nachfolgende Abbildung 16 verdeutlicht diesen Sachverhalt.

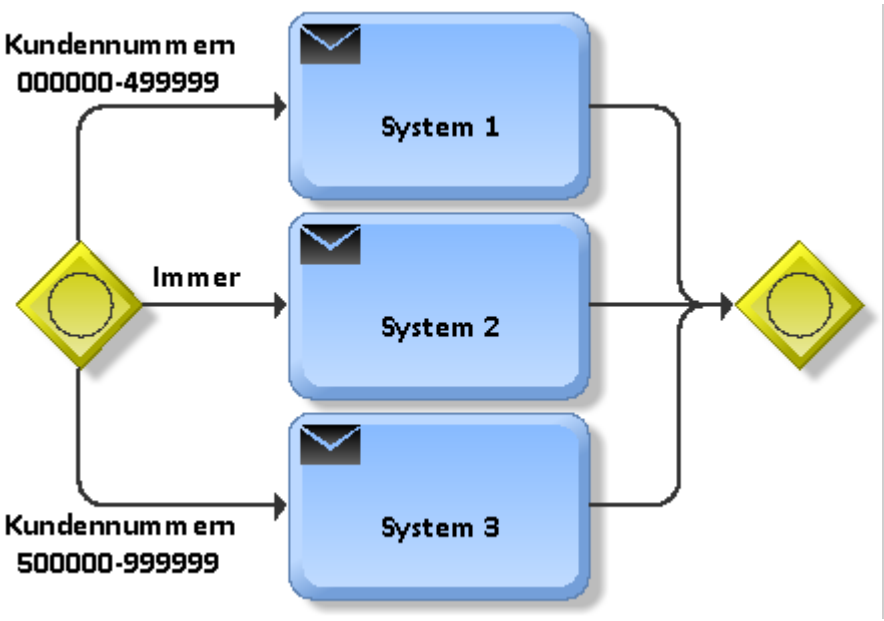



*Abbildung 16: Inklusives Gateway mit Pfad ohne Bedingung*

Mit System 2 ist demnach immer zu kommunizieren, während mit den beiden anderen Systeme nur dann Nachrichten ausgetauscht werden, wenn die Bedingung für den jeweiligen Kundennummernkreis erfüllt ist.

In vielen Szenarien muss auf das Eintreffen von Nachrichten gewartet werden. Statt diese Wartezeiten ungenutzt verstreichen zu lassen, können parallel weitere Aktivitäten durchgeführt werden. Das in Abbildung 17 dargestellte Modell veranschaulicht ein mögliches Szenario.

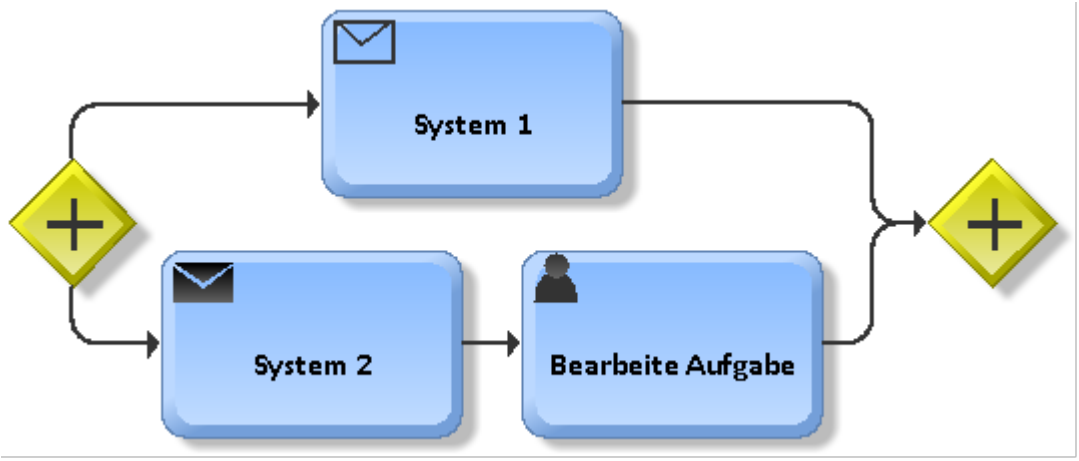



*Abbildung 17: Nutzung von Wartezeiten durch Verwendung eines parallelen Gateways*

Während der Wartezeit auf eine Nachricht von System 1 wird parallel mit einem zweiten System kommuniziert und anschließend eine Benutzeraufgabe abgewickelt. Im darauffolgenden parallelen Gateway werden die beiden Stränge schließlich wieder synchronisiert.

Als weitere Möglichkeit zur Modellierung von Nebenläufigkeit in BPMN ist die Verwendung der nicht-unterbrechenden angehefteten Zwischenereignisse zu nennen. Ein ähnliches Verhalten ist auch mit Ereignis-Unterprozessen zu erreichen. Beide Möglichkeiten sollen im Folgenden dargestellt werden.

Die nicht-unterbrechenden angehefteten Zwischenereignisse werden verwendet, wenn während der Bearbeitung einer Aufgabe oder eines Unterprozesses Ereignisse eintreten, die zwar behandelt werden müssen, die allerdings nicht so schwerwiegend sind, als dass dafür die aktuell laufende Bearbeitung unterbrochen werden müsste. So kann z.B. während der Buchung einer Reise der Kunde eine neue Kreditkarte im System registrieren, die nach erfolgreicher Buchung zu belasten ist. Nur weil der Kunde die neuen Informationen bereitstellt, soll die eigentliche Buchung nicht unterbrochen werden (vgl. auch OMG 2010b). Unter Verwendung von angehefteten nicht-unterbrechenden Zwischenereignissen ergibt sich das in Abbildung 18 dargestellte Modell.

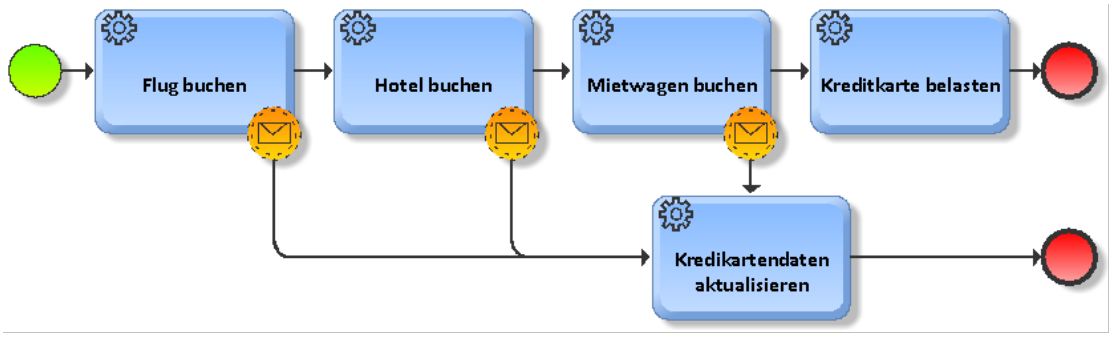



*Abbildung 18: Einsatz nicht-unterbrechender Ereignisse (in Anlehnung an OMG 2010b, S. 28)*

Die Nachricht über die geänderten Kreditkartendaten kann während der Bearbeitung jeder der drei Buchungsaufgaben eintreten. Entsprechend muss das nicht-unterbrechende Zwischenereignis an jeden der drei Aufgaben hinzugefügt werden. Eleganter lässt sich der Sachverhalt über ein nicht-unterbrechendes Zwischenereignis modellieren, dass an einen Unterprozess geheftet ist. Der Unterprozess wiederum umfasst genau die drei Aufgaben, die die eigentliche Reisebuchung ausmachen. Das Ergebnis ist in Abbildung 19 zu sehen.

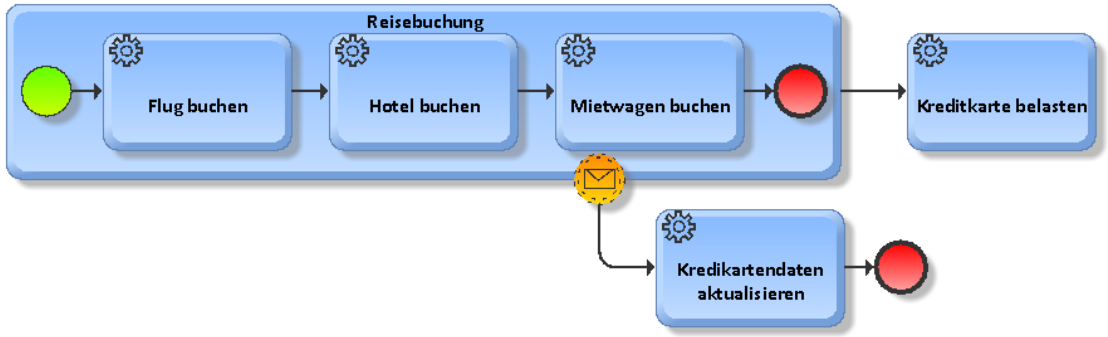



***Abbildung 19: Einsatz nicht-unterbrechender Ereignisse an einem Teilprozess (in Anlehnung an OMG 2010b, S. 28)***

Durch die Verwendung eines Unterprozesses wird zudem gleichzeitig der Gültigkeitsbereich für das Event festgelegt: das Eintreffen des Ereignisses wird nur während der Bearbeitung der drei Buchungsaufgaben gestattet. Wird die Kreditkarte hingegen bereits belastet, ergibt eine Kreditkartendatenaktualisierung keinen Sinn mehr.

Dasselbe Szenario umgesetzt mit den in BPMN 2.0 neu eingeführten Ereignis-Unterprozessen ist in Abbildung 20 dargestellt.

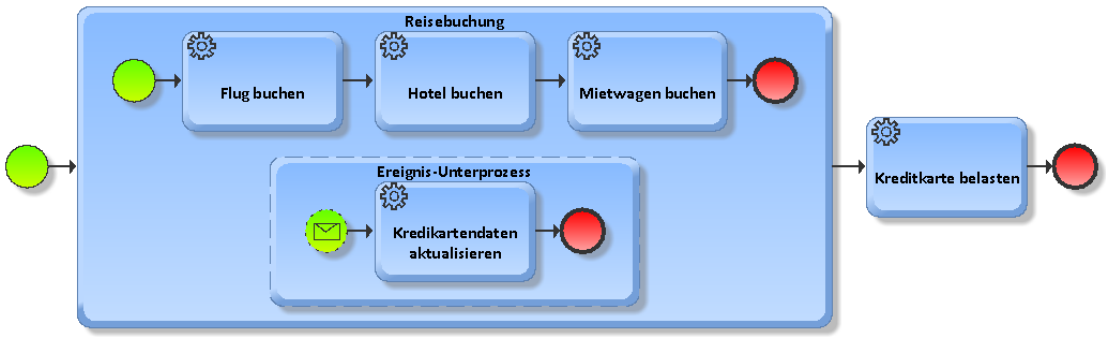



***Abbildung 20: Einsatz eines Ereignis-Unterprozesses mit nicht-unterbrechendem Startereignis (in Anlehnung an OMG 2010b, S. 28)***

Der Ereignis-Unterprozess ist unterhalb des Reisebuchungsprozesses zu erkennen. Charakteristisch ist dessen gestrichelter Rand. Er ist während der Durchführungsdauer des Hauptprozesses aktiv und wird durch die eingehende Nachricht aktiviert. Da das Startereignis im Unterprozess nicht-unterbrechend ist (gestrichelter Rand des Startereignisses), findet die Kreditkartendatenaktualisierung wiederum parallel zur Ausführung des Buchungssequenzflusses statt. Auch wenn die beiden Modelle aus Abbildung 19 und Abbildung 20 sich sehr ähnlich sind und auf den ersten Blick keine großen semantische Unterschiede erkennbar sind, so verhalten sich die beiden Modelle zur Laufzeit dennoch verschieden: bei der Verwendung des Ereignis-Unterprozesses (Abbildung 20) wird die Beendigung des Reisebuchungs-Unterprozesses solange verzögert, bis auch der Ereignisunterprozess beendet ist. Dieses Verhalten wird durch die Einbettung des Ereignis-Unterprozess in den Reisebuchungs-Unterprozess impliziert. Von daher sollten keine zeitaufwändigen Aktivitäten in Ereignis-Unterprozesse eingebettet werden, da sonst die Gefahr besteht, dass der weitere Prozessablauf unnötig lang verzögert wird.

Ganz anders hingegen das Verhalten des angehefteten nicht-unterbrechenden Ereignisses (Abbildung 19): hier befindet sich die Ereignisbehandlung außerhalb des Unterprozesses, also muss nicht auf dessen Beendigung gewartet werden, um das Token aus dem Unterprozess an den ihm übergeordneten Prozess weiterzureichen. Es wird also ein noch höherer Parallelisierungsgrad erreicht: die Ereignisbehandlung läuft sowohl parallel zum Unterprozess als auch zum übergeordneten Prozess ab. Diese Unterscheidung muss dem Modellierer bewusst sein, um je nach Situation das richtige Muster zu verwenden.

Alle drei Darstellungen ermöglichen folglich eine recht kompakte Darstellung von Nebenläufigkeit in BPMN-modellierten Prozessen.

Neben den Vorteilen derartiger Parallelisierungen muss sich der Modellierer allerdings auch seiner besonderen Verantwortung hinsichtlich der prozessinternen Datenbehandlung und -zugriffe bewusst sein. Schließlich arbeiten die einzelnen Aufgaben mit den Daten des Prozesskontextes, der nur einmalig vorhanden ist. Bei den parallelen Zweigen ist daher der konkurrierende Zugriff auf den Prozesskontext zu berücksichtigen. Gegebenenfalls sind Maßnahmen zu ergreifen, die ein Überschreiben verhindern. Dies kann beispielsweise durch getrennte Bereiche im Prozesskontext erzielt werden.

### *4.1.3 Ausnahmebehandlung*

Während der Durchführung von Aufgaben, Unterprozessen und Prozessflusssequenzen wird es immer wieder zu Fehlersituationen kommen. Dafür passende Sprachmittel zur Verfügung zu stellen, um geeignet reagieren zu können, zeichnet gute Notationen aus. Auch hier liefert BPMN geeignete Konstrukte, die dem Modellierer das Leben vereinfachen. Einige Beispiele verdeutlichen diesen Aspekt. So ist in Abbildung 21 die grundsätzliche Unterscheidung zwischen verschiedensten Fehlerursachen dargestellt. Dabei wird der Unterprozess *Reisebuchung* mit der Sequenz *Buchungsdaten senden* und *Buchungsbestätigung empfangen* aus Sicht der Ausnahmebehandlung als Einheit betrachtet. Nun sind folgende Fehlerfälle denkbar: der Empfänger der Buchungsdaten stellt eine betriebswirtschaftliche Inkonsistenz fest und zeigt dies durch eine entsprechende Ausnahme an.

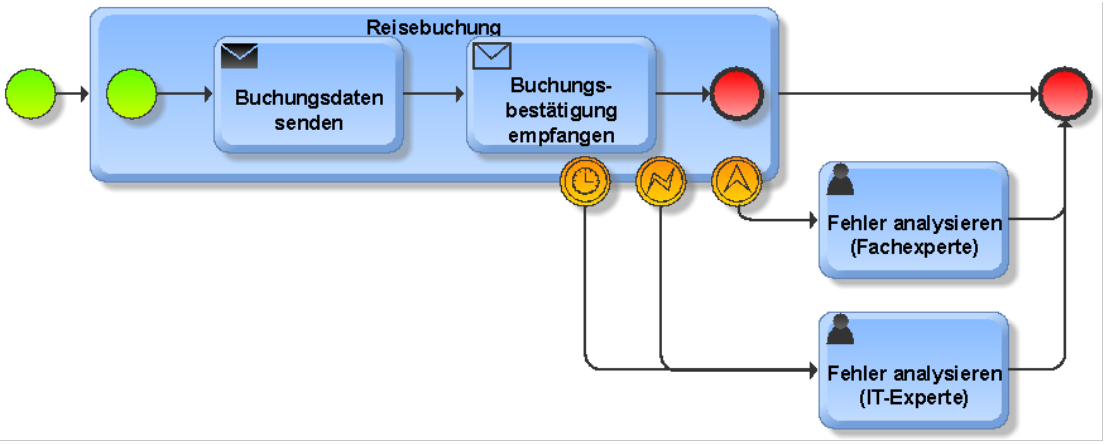



*Abbildung 21: Fehlerbehandlung durch Verwendung unterschiedlicher Ereignisse*

Zur Behandlung eignet sich in der BPMN die Eskalation (), die fachliche Probleme signalisiert. Bei technischen Fehlern wie Datenbankproblemen oder Ausfall der Netzwerkverbindung ist das Fehler-Ereignis () zu nehmen. Gleichzeitig wird in dem Modell der Unterprozess zeitlich überwacht. Erreicht die Buchungsbestätigung nicht innerhalb einer vorgegebenen Zeit die Empfangsaufgabe, führt auch dies zu einer Ausnahmebehandlung, die über das Zeitereignis () abgefangen wird. Durch das Anheften der Ereignisse wird gleichzeitig der Gültigkeitsbereich der Fehlerüberwachung festgelegt. In obigen Beispiel gilt sie für die Dauer der Unterprozessausführung. Dieselbe Wirkung erzielen auch drei in den Unterprozess eingebettete Ereignis-Unterprozesse. Da es sich diesmal um unterbrechende Ereignisse handelt, ist die Ausführung in der Tat identisch zu den an den Unterprozess angehefteten Ereignissen. Die Ursache dafür ist, dass die Ereignisbearbeitung nicht parallel zum Unterprozess ausgeführt wird, sondern alternativ.

Genauso ist es möglich, die Ereignisse an eine einzelne Aufgabe anzuheften. Dann beschränkt sich die Überwachung ausschließlich auf die aktuelle Task.

Auf diese Weise lassen sich flexibel unterschiedliche Fehlerbehandlungen für die verschiedenen Fehlertypen modellieren. Dies ist insbesondere beim Zusammenspiel zwischen Verbundanwendung und Servicevertrag-Implementierungsschicht von Bedeutung: technische Fehlerursachen können in der Implementierungsschicht behandelt werden, während fachliche Fehler an die Composite durchzureichen sind.

In BPMN ist aber nicht nur eine Unterscheidung zwischen den Fehlertypen möglich. Auch innerhalb eines Fehlertyps können wiederum diverse Zustände unterschieden werden, wie dies in Abbildung 22 dargestellt ist.

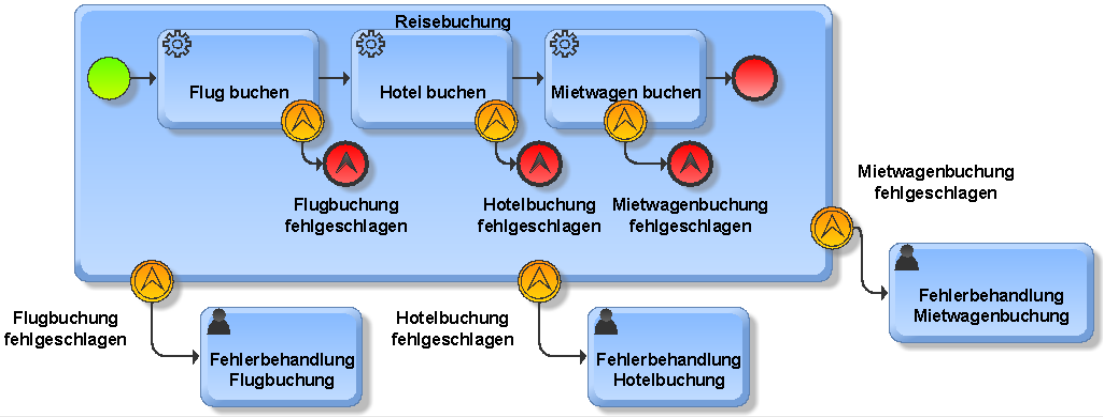



*Abbildung 22: Unterscheidung verschiedener Fehlerzustände mit dem Eskalationsereignis*

In diesem Beispiel wurden drei unterschiedliche fachliche Fehlerursachen modelliert, dargestellt durch drei Eskalationsereignisse, die jedoch verschieden annotiert wurden. Dadurch ist eine eindeutige Zuordnung zu den Fehlerbehandlungen möglich. Für Verbundanwendungen wiederum bedeuten diese Möglichkeiten eine Vielfalt von Implementierungsalternativen zur Ausnahmebearbeitung.

#### 4.1.3.1 Zeitüberwachung

Die Zeitüberwachung nimmt innerhalb der Ausnahmebearbeitung eine Sonderstellung ein, da mit Hilfe der Timer-Ereignisse eine Vielzahl von zeitlichen Abhängigkeiten modelliert und durch entsprechende Laufzeitumgebungen dann auch zur Ausführung gebracht werden können. Bei der Spezifikation von Verbundanwendungen wurde bereits der Aspekt der zeitlichen Vorgabe für den Servicevertrag angesprochen: in ihr wird definiert, ob die Ausführung des Servicevertrags in einer bestimmten Zeit abzuwickeln ist und wenn ja, wie die Reaktionen bei Überschreitungen auszusehen haben. Die folgenden Beispiele verdeutlichen die Möglichkeiten, die BPMN hier zu bieten hat.

Das erste Beispiel adressiert ein typisches Szenario in der Servicevertrag-Implementierungsschicht: es soll die Ausführungszeit für den Empfang einer Antwortnachricht überwacht werden. Die Composite Application hat einen bestimmten Service angefragt und erwartet eine Antwort innerhalb einer bestimmten Zeit. Trifft die Antwort nicht innerhalb dieses Zeitraums ein, so soll der Vorgang abgebrochen und eine Feh-

lermeldung an den Aufrufer zurückgeschickt werden. Abbildung 23 zeigt ein dazu passendes BPMN-Modell.

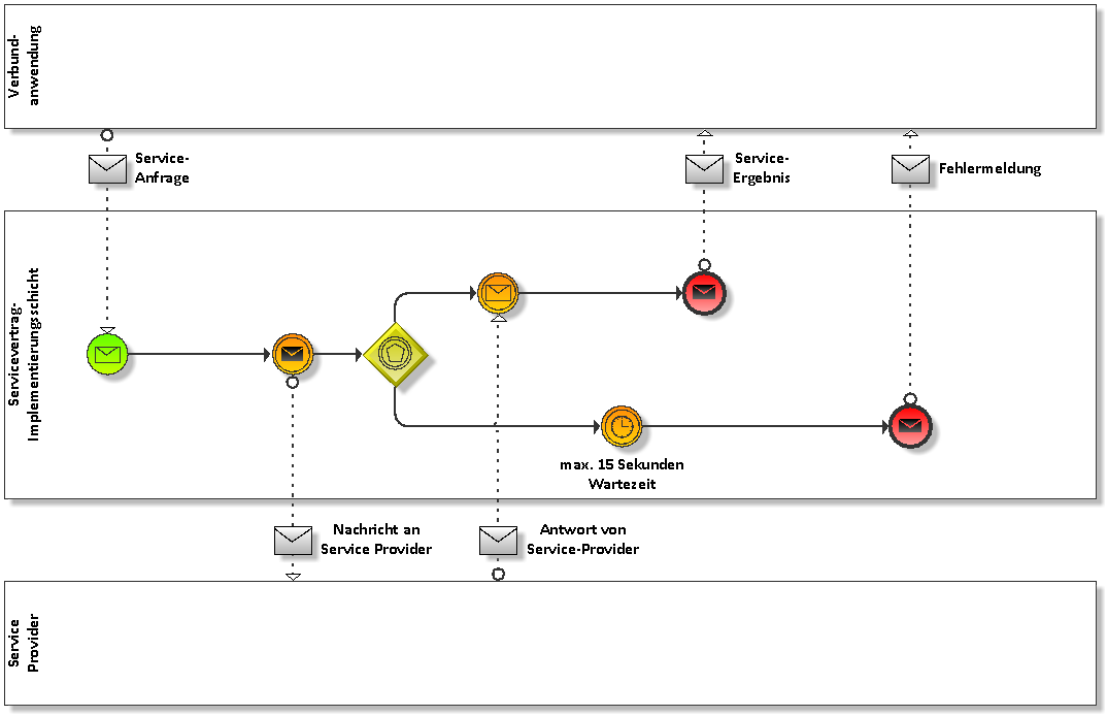



***Abbildung 23: Zeitüberwachung des Nachrichtenempfangs mit Hilfe des ereignisbasierten Gateways***

Aus der Verbundanwendung heraus wird die Service-Anfrage verschickt. Der Prozess wird daraufhin über das Nachrichten-Startereignis instanziiert und verschickt als erste Aktion seinerseits eine Nachricht an den Service-Provider (zu erkennen an dem sendenden Nachrichten-Zwischenereignis). Das durch den Prozessstart instanziierte Token erreicht nun das ereignisbasierte Gateway. Hier verbleibt es so lange, bis eines der beiden nachfolgenden Ereignisse eintritt: entweder trifft die Antwortnachricht des Service-Providers ein und der Prozess wird über den oberen Pfad und dem damit verbundenen Versand des Ergebnisses beendet, oder die maximale Wartezeit von 15 Sekunden läuft ab und die Verbundanwendung wird über eine entsprechende Fehlermeldung über diesen Vorfall informiert. Auf diese Weise wird eine maximale Antwortzeit von 15 Sekunden überwacht.

Die Modellierung der Zeitüberwachung mittels des ereignisbasierten Gateways ist ein beliebtes Mittel zur Umsetzung derartiger Anforderungen. Allerdings ist sie auch mit zwei Nachteilen verbunden:

1. Auf diese Weise können lediglich Ereignisse überwacht werden, jedoch keine Aktivitäten.
2. Welches Ereignis auch immer eintrifft, es ist stets unterbrechend.

Für die Lösung des ersten Problems eignen sich angeheftete Ereignisse. Dasselbe Szenario wie aus Abbildung 23 modelliert mit Aktivitäten und einem angehefteten Zeitereignis ist in Abbildung 24 dargestellt.

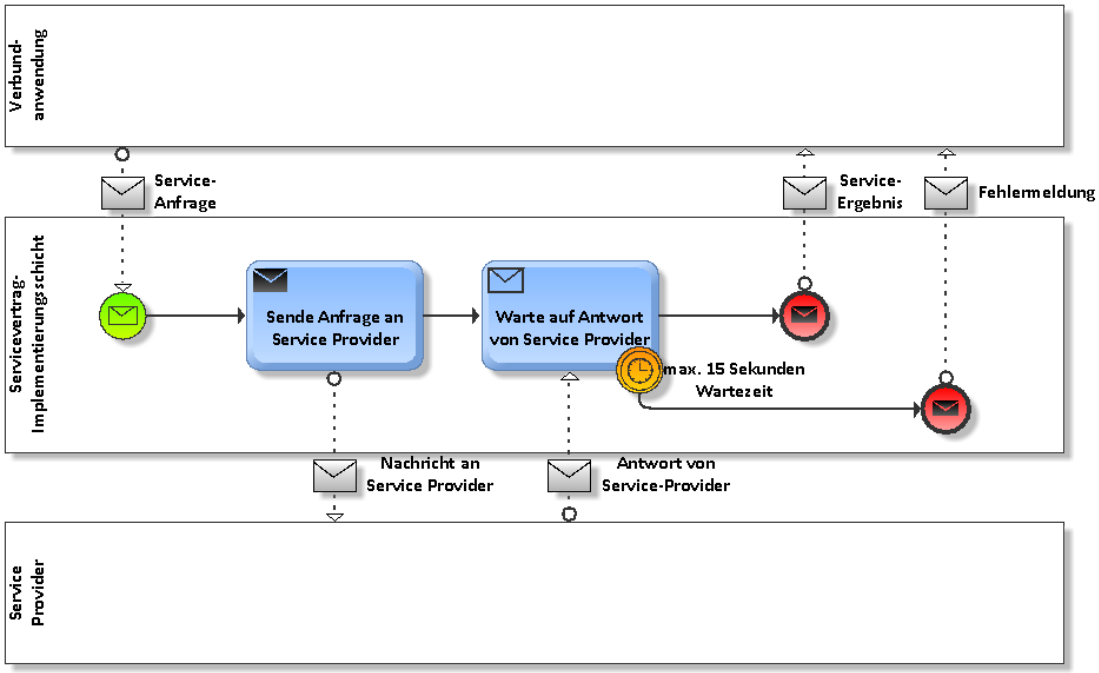



*Abbildung 24: Zeitüberwachung mit angeheftetem Zeitereignis (unterbrechend)*

Hier wird die Empfangsaufgabe mit einem angehefteten unterbrechenden Zeitereignis versehen und dadurch die Zeitüberwachung zum Ausdruck gebracht. Genauso gut hätte jetzt jede andere Aufgabe, wie beispielsweise eine Benutzer- oder eine Geschäftsregelaufgabe verwendet werden können. Diese Art der Modellierung bietet aber auch gleichzeitig für das zweite Problem eine Lösung, da das unterbrechende Zeitereignis ebenso durch ein nicht-unterbrechendes Ereignis ersetzt werden kann. Ist beispielsweise gewollt, dass nach 15 Sekunden lediglich ein Administrator informiert werden soll, damit dieser sich parallel zum weiteren Warten auf das Eintreffen einer Antwort mit dem Problem auseinandersetzen kann, so sieht das Modell wie in Abbildung 25 dargestellt aus.

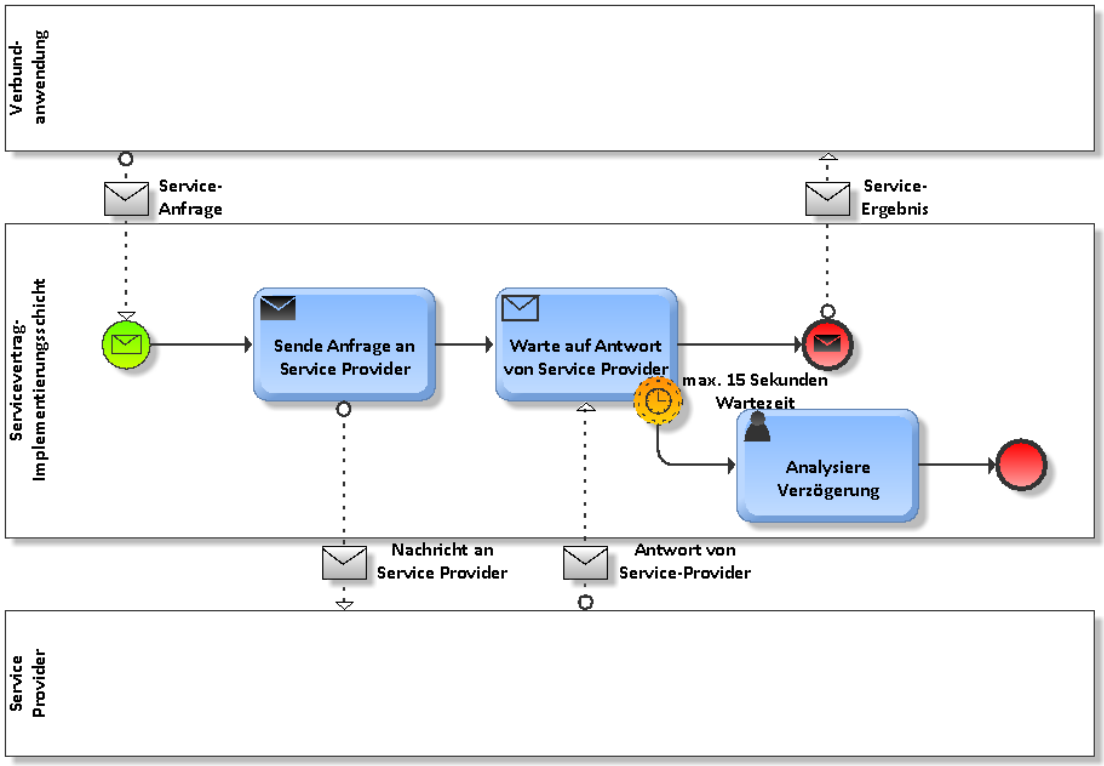



*Abbildung 25: Zeitüberwachung mit angeheftetem Zeitereignis (nicht-unterbrechend)*

In der Abbildung ist das Eingreifen des Administrators nach Ablauf der vereinbarten Zeit zu erkennen. Auch wenn sein Sequenzfluss in einem Endereignis mündet, so ist dadurch der Prozess noch nicht abgeschlossen, da sein Pfad durch ein nicht-unterbrechendes Ereignis ausgelöst wurde. Folglich befindet sich noch immer ein Token auf der Empfangs-Aufgabe, der Prozess ist also nach wie vor aktiv.

Die Kombination aus Modellierung von Sequenzflüssen mit Aufgaben und dem Anheften von unterbrechenden bzw. nicht-unterbrechenden Ereignissen gibt dem Modellierer den größten Handlungsspielraum. Zusätzlich eröffnen sich ihm durch die Verwendung von Teilprozessen, die ihrerseits wieder zeitüberwacht sind, weitere Möglichkeiten. Eine geeignete Prozessstrukturierung vorausgesetzt, erstrecken sich die Kontrolloptionen dann von einzelnen Aufgaben über Teilsequenzen bis hin zu Gesamtprozessen. Als Beispiel veranschaulicht die nachfolgende Abbildung 26 die Überwachung eines Teilprozesses. So ließe sich die Anforderung umsetzen, dass die Antwort innerhalb von 5 Minuten vorliegen muss. In diesem Fall genügt eine Einzelschrittüberwachung nicht, sie muss sich jetzt über mehrere Aktivitäten erstrecken.

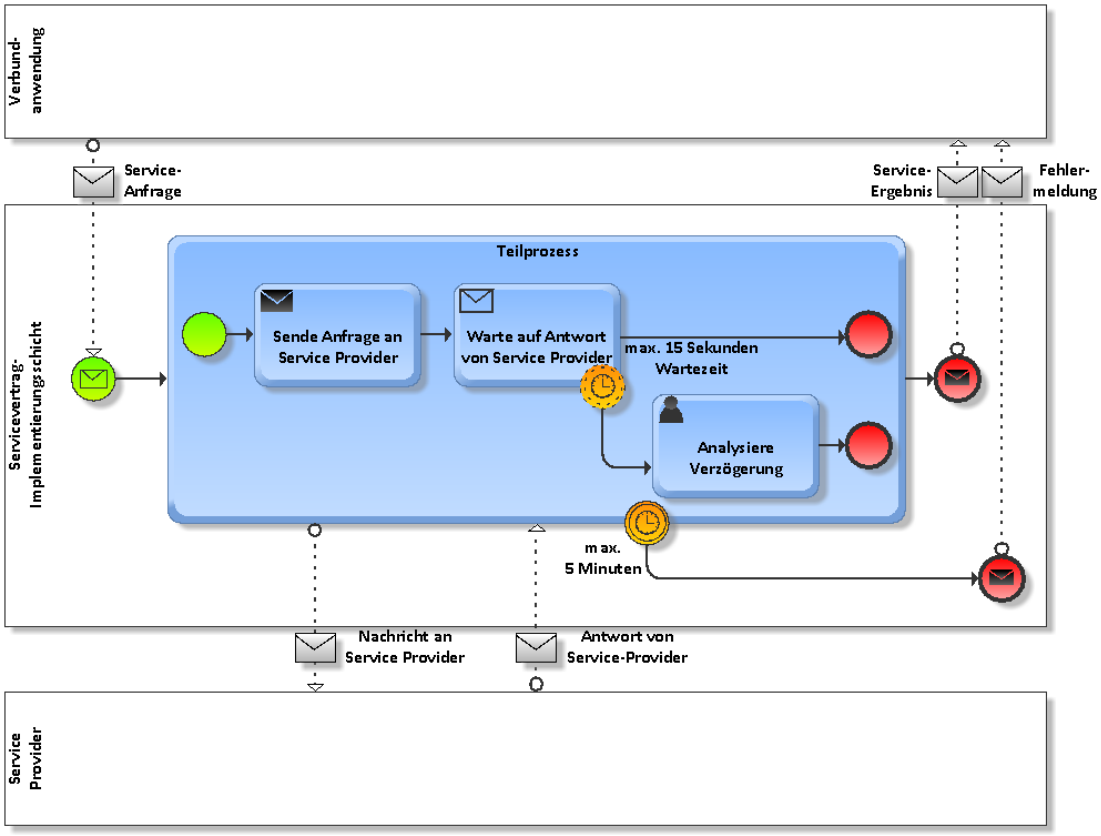



*Abbildung 26: Zeitliche Überwachung eines Teilprozesses*

Wiederum ist die Zeitüberwachung der Empfangsaufgabe zu erkennen. Darüber hinaus befindet sich die gesamte Nachrichtenbehandlung innerhalb eines Teilprozesses, an dem sich ein unterbrechendes Zeitereignis befindet, das nach fünf Minuten die gesamte Kommunikation unterbindet und mit einer Fehlermeldung die Verbundanwendung über den missglückten Serviceaufruf informiert.

Als letzte Variante zur Modellierung von Zeitüberwachungen wird in einem weiteren Beispiel die Verwendung von Ereignisunterprozessen veranschaulicht. Ereignisunterprozesse sind während der Laufzeit eines Prozesses oder eines Teilprozesses aktiviert und bereit, auftretende Ereignisse zu behandeln. Diese können wiederum unterbrechend oder nicht-unterbrechend sein. Zudem zeigt das BPMN-Modell aus Abbildung 27 eine kaskadierte Zeitüberwachung: es können maximal drei Zeitereignisse eintreten.

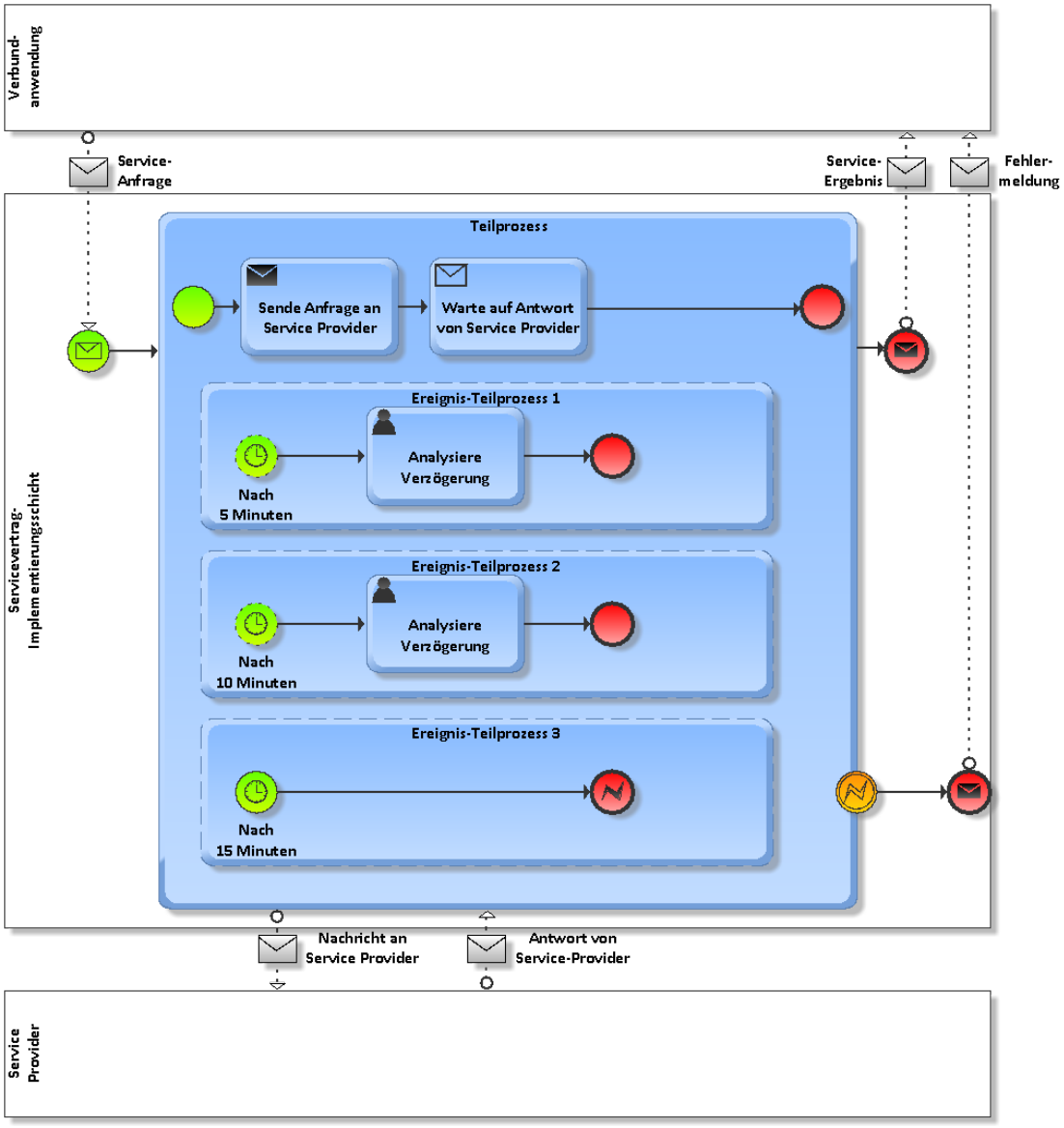



*Abbildung 27: Zeitüberwachung durch Verwendung von Ereignis-Teilprozessen*

Der eigentliche Teilprozess behandelt den Nachrichtentransfer von und zu dem Service-Provider. Sollte innerhalb von 5 Minuten keine Nachricht eingetroffen sein, greift das erste nicht-unterbrechende Zeitereignis, woraufhin Ereignis-Teilprozess 1 gestartet und ein Administrator über das Problem informiert wird. Nach weiteren 5 Minuten wird Ereignis-Teilprozess 2 wiederum über ein nicht-unterbrechendes Zeitereignis gestartet. Ist auch nach insgesamt 15 Minuten noch immer keine Nachricht eingetroffen, sorgt Ereignis-Teilprozess 3 durch sein unterbrechendes Startereignis für die Beendigung der Kommunikation. Durch das Fehler-Endereignis wird dem umge-

benden Teilprozess die Fehlersituation angezeigt, woraufhin dieser mit einer Fehlermeldung an die aufrufenden Composite geeignet reagieren kann.

Diese Beispiele zeigen die flexiblen Möglichkeiten, mit BPMN vielfältige Zeitüberwachungsszenarien abzubilden. Dabei wurde zugleich Wert auf die Relevanz derartiger Prozesse für Verbundanwendungen im Zusammenspiel mit ihrer Servicevertrag-Implementierungsschicht gelegt.

### *4.1.4 Kollaborationen und Nachrichtenaustausch*

In den Prozessmodellen des vorangegangenen Abschnitts fällt die Verwendung mehrerer Pools auf: ein Pool repräsentiert die Verbundanwendung, ein anderer die Servicevertrag-Implementierungsschicht und ein dritter den Service Provider. Zwischen den Pools wird durch Pfeile der Nachrichtenfluss dargestellt. Auf diese Weise wird das Zusammenspiel der Prozesse veranschaulicht. In der BPMN-Nomenklatur wird diese Darstellung als *Kollaborationsdiagramm* bezeichnet. Das Kollaborationsdiagramm eignet sich hervorragend zur Modellierung der Interaktionen zwischen dem fachlichen und dem technischen Prozess. Auch für die verschiedenen Arten der Kommunikation zwischen den Kollaborationspartnern sieht die BPMN unterschiedliche Elemente vor. So kann die synchrone Kommunikation unter Verwendung der Service-Aufgabe, die asynchrone Kommunikation unter Verwendung von sendender und empfangender Aufgabe, sowie Publish-Subscribe-Szenarien unter Verwendung von Signalen modelliert werden. Die dazugehörigen Modelle sind in den Abbildungen 28, 29 und 30 abgebildet.

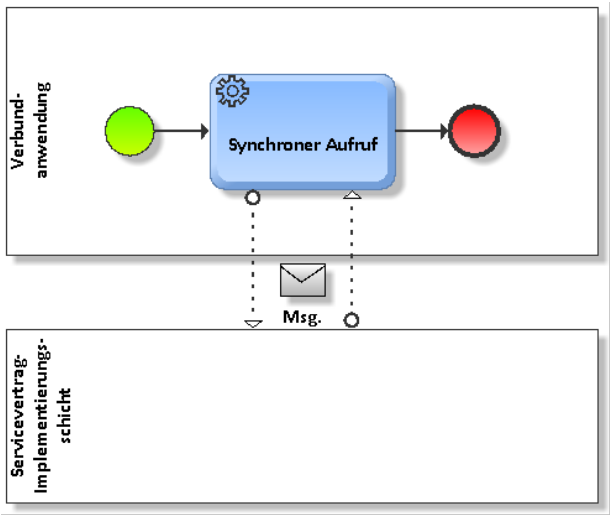



*Abbildung 28: Synchrone Kommunikation unter Verwendung der Service-Aufgabe*

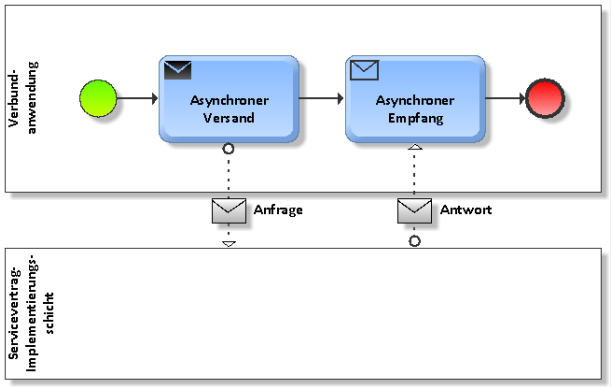



*Abbildung 29: Asynchrone Kommunikation unter Verwendung von sendender und empfangender Aufgabe*

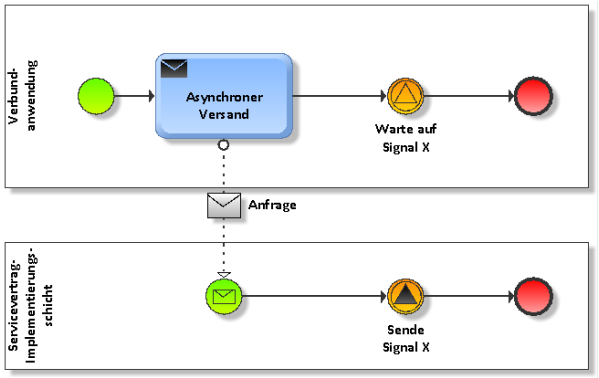



*Abbildung 30: Asynchrone Kommunikation unter Verwendung von Signalen*

Neben den Kollaborationsdiagrammen wurden in BPMN 2.0 noch zwei weitere Diagrammtypen zur Darstellung von Kommunikationsvorgängen eingeführt: das *Choreographie-* und das *Konversationsdiagramm* (siehe OMG 2010a und Allweyer 2009b). Das Choreographiediagramm repräsentiert die Interaktionen zwischen zwei Pools (siehe auch Allweyer 2009b, S. 13ff und Freund, Rücker, Henninger 2010, S.143/144), wobei im Gegensatz zum Kollaborationsdiagramm auch logische Zusammenhänge zwischen den Nachrichtenaustauschen dargestellt werden. Bei Kollaborationsdiagrammen wird lediglich dargestellt, *dass* eine Kommunikation stattfindet. Bei Choreographiediagrammen kann zudem entnommen, *wann welche Nachricht unter welchen Umständen* verschickt und empfangen wird.

Die Konversationsdiagramme fassen Nachrichtenaustausche zusammen. Dazu müssen zwischen den Nachrichtenaustauschen über Korrelationen Beziehungen bestehen. Diese werden dann als logisch zusammengehörige Austausche bezeichnet. So besteht beispielsweise zwischen einer Bestellung und der dazugehörigen Bestellbestätigung über die Bestellnummer eine eindeutige Beziehung. Auch Stornierung und Lieferverzug beziehen sich auf eine konkrete Bestellung. All diese Nachrichtenflüsse können im Konversationsdiagramm dargestellt werden.

Beide Diagrammtypen spielen im Rahmen dieser Arbeit keine Rolle, so dass sie auch nicht weiter vertieft werden. Weitere Details speziell zu den neuen Diagrammtypen können u.a. Allweyer 2009b entnommen werden.

### *4.1.5 Transaktionen und Kompensation*

Die Servicevertrag-Implementierungsschicht muss bei der Umsetzung der von der Verbundanwendung angeforderten Dienste die transaktionale Integrität gewährleisten. Dazu stehen ihr in der BPMN verschiedenste Konstrukte zur Verfügung. Speziell der Transaktions-Teilprozess wurde für diese Aufgabe konzipiert. Seine Funktionsweise lässt sich anhand des Beispiels aus Abbildung 31 erläutern.

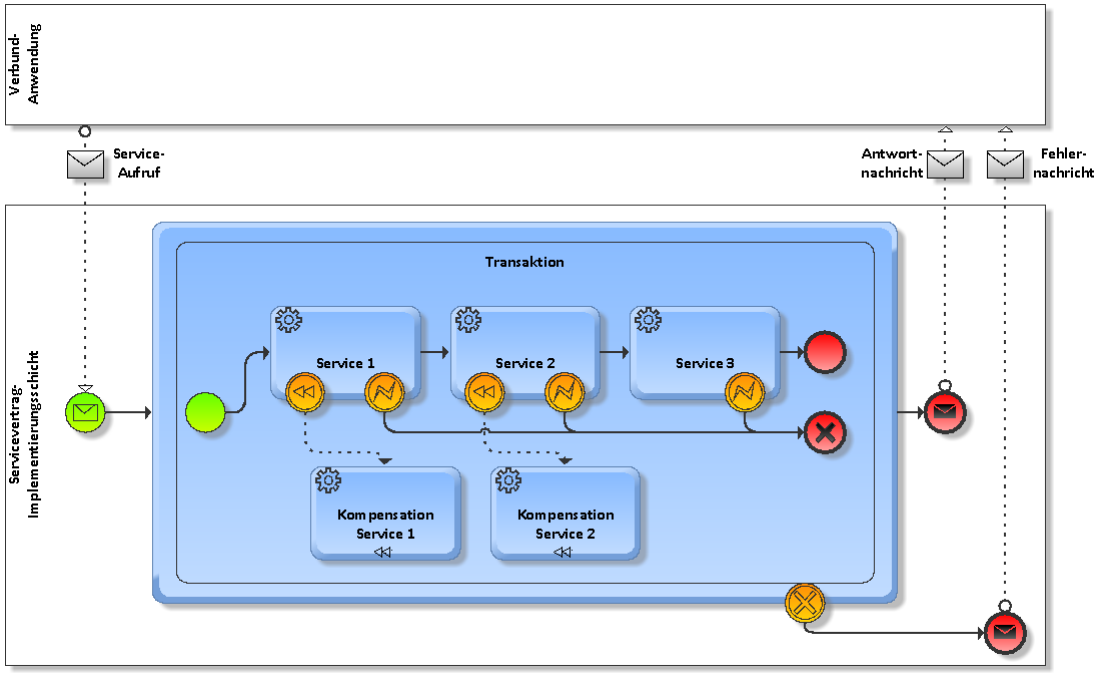



*Abbildung 31: Transaktions-Teilprozess*

Die Verbundanwendung setzt einen Service-Aufruf zum Schreiben von Daten ab. Diese müssen in drei Systemen verbucht werden. Entweder alle drei Schreibzugriffe sind erfolgreich oder die Transaktion ist gescheitert und die bereits erfolgreich abgeschlossenen Aufrufe müssen wieder durch Kompensation zurückgerollt werden. Die drei Schreibzugriffe sind als synchrone Service-Aufgaben innerhalb der Transaktion zusammengefasst. Transaktionen erkennt man in BPMN an der doppelten Umrandung. Bei allen drei Aufrufen fällt das unterbrechende angeheftete Fehlerereignis auf. Bei dessen Eintreten wird innerhalb der Transaktion das Abbruch-Endereignis erreicht (⊗). Dieses Endereignis sorgt für die Aktivierung der kompensierenden Sequenzflüsse, die ebenfalls den Service-Aufgaben angeheftet wurden. Sie werden allerdings nur dann ausgeführt, wenn die dazugehörige Aufgabe zuvor auch korrekt beendet wurde. Die Kompensationen werden dabei in umgekehrter Reihenfolge aufgerufen. Konkret bedeutet das bei einem Fehler im letzten Schritt (Service 3), dass zunächst Service 2 und erst danach Service 1 zurückgerollt würden. Nach Abschluss der Kompensationen wird dem an der Transaktion angehefteten Abbruch-Zwischenereignis gefolgt und eine Fehlermeldung an die Verbundanwendung zurückgeschickt.

Zusammenfassend lässt sich sagen, dass BPMN zur Abwicklung von Transaktionen das Zusammenspiel von fünf BPMN-Elementen vorsieht:

- Transaktion: eine spezielle Aktivität, die die als eine Transaktion auszuführenden Schritte enthält.
- Abbruch-Endereignis: wird innerhalb der Transaktion erreicht, sobald bei der Ausführung einer der in der Transaktion befindlichen Schritte ein Fehler festgestellt wird und damit die eigentliche Kompensation auslöst.
- Abbruch-Zwischenereignis: ist an der Transaktion angeheftet und legt den Sequenzfluss fest, dem unmittelbar nach der Kompensationsbehandlung zu folgen ist.
- Kompensations-Zwischenereignis: wird an die an einer Transaktion beteiligten Aufgaben geheftet und ist mit der zu einer Aufgabe gehörenden...
- ...Kompensationsaufgabe verbunden. Die Kompensationsaufgabe ist mit einer Kompensationsmarkierung (⏪) versehen und wird während der Rollback-Verarbeitung aufgerufen. Ihre Aufgabe ist es, die während des normalen Ablaufs abgeschlossene fachliche Veränderung wieder zu kompensieren. Kompensations-Zwischenereignis und Kompensationsaufgabe sind über eine Assoziation miteinander verbunden.

Doch nicht nur während der Transaktionsverarbeitung ist das Zurückrollen von Aufgaben möglich. Selbst vollständig abgeschlossenen Transaktionen können nachträglich noch zurückgerollt werden, wie dies in Abbildung 32 zu sehen ist.

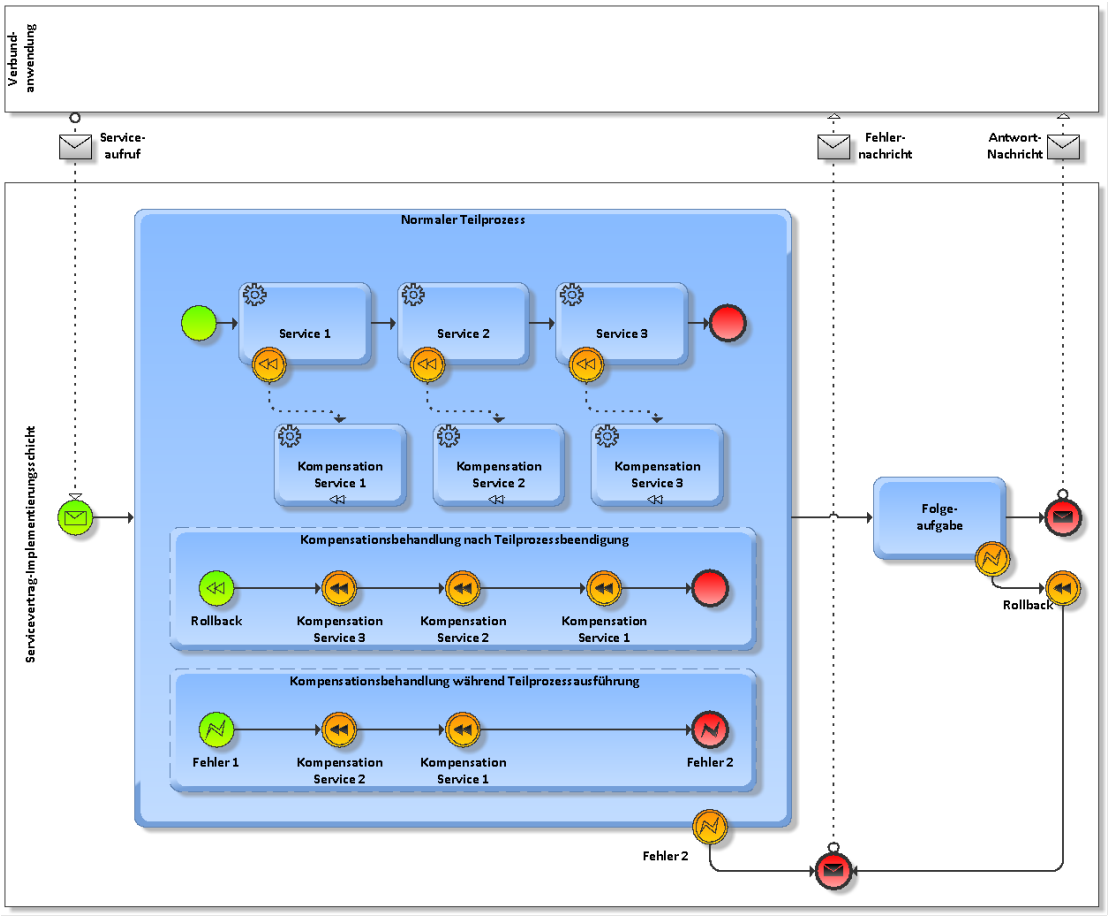



***Abbildung 32: Zurückrollen erfolgreich abgeschlossener Transaktionen***

Die als eine Transaktion zu behandelnden Service-Aufgaben sind diesmal in einem normalen Teilprozess eingebettet. Allerdings werden sie zur Prozessausführungszeit durch einen Ereignis-Teilprozess überwacht. Er ist in der Abbildung mit *Kompensationsbehandlung während Teilprozessausführung* bezeichnet und wird angesprungen, sobald während der Teilprozessausführung ein Fehler auftritt. In ihm sind die kompensierenden Aufrufe durch auslösende Kompensations-Zwischenereignisse explizit modelliert worden. Zur Vereinfachung wurde angenommen, dass die aufgerufenen Kompensationsaufgaben selbstständig erkennen, ob eine Kompensation notwendig ist oder nicht. Wäre dies nicht der Fall, so hätte vor den Kompensationsaufgaben jeweils ein exklusives Gateway den Aufruf der Kompensation nur im Falle einer zuvor erfolgreichen Ausführung sicherstellen müssen. Dies hätte das Prozessmodell allerdings nur unnötig verkompliziert. Nach Beendigung des Ereignis-Teilprozesses wird ein weiterer Fehler (*Fehler 2*) ausgelöst, damit der umgebende Hauptprozess eine Fehlernachricht an den Aufrufer absetzen kann.

Tritt während der eigentlichen Teilprozessausführung kein Fehler auf, so wird die Aufgabe *Folgeaufgabe* erreicht. Wird auch sie erfolgreich beendet, kann der Hauptprozess mit dem Versand der Antwortnachricht beendet werden. Wird hingegen bei ihrer Ausführung ein Fehler festgestellt, so wird dem angehefteten Fehlerereignis folgend das auslösende Kompensationsereignis *Rollback* erreicht. Dieses bewirkt nun nachträglich ein Rollback aller in dem Teilprozess abgeschlossener Aufgaben, indem der in dem Teilprozess enthaltene Kompensations-Ereignis-Teilprozess namens *Kompensationsbehandlung nach Teilprozessbeendigung* aufgerufen wird. Er ist an dem empfangenden Kompensations-Startereignis zu erkennen. Die nachfolgenden auslösenden Kompensations-Zwischenereignisse bewirken den Aufruf der jeweiligen Kompensationsaufgaben. Wurden sämtliche Aufgaben zurückgerollt, kann der Hauptprozess wiederum durch Versand einer Fehlermeldung beendet werden.

Einmal mehr verdeutlichen diese Beispiele die vielfältigen Modellierungsmöglichkeiten unter Verwendung der BPMN, von denen insbesondere die Implementierungen von Verbundanwendungen und ihrer dazugehörigen Servicevertrag-Implementierungsschicht profitieren. Von daher wird in der nun folgenden konkreten Implementierung BPMN für beide Schichten eingesetzt. Damit unterscheidet sich das in dieser Arbeit beschriebene Vorgehen von Ansätzen, die noch auf die ereignigesteuerten Prozessketten (EPK) zur fachlichen Modellierung und auf BPEL zur technischen Umsetzung zurückgreifen (siehe auch Stein 2009). Zum Vergleich von BPMN mit EPK sei auf Stein 2010 verwiesen.

## 4.2 Beispielimplementierung der Grundarchitektur einer Verbundanwendung als Proof-of-Concept

In diesem Abschnitt wird anhand eines konkreten Beispiels die Umsetzung einer Verbundanwendung einschließlich der dazugehörigen Servicevertrag-Implementierungsschicht erläutert. Die Implementierung erfolgt dabei mittels der Modellierungs- und Ablaufumgebung SAP NetWeaver Composition Environment (SAP NetWeaver CE) in seiner aktuellsten Version 7.2. In einem kurzen Abschnitt wird diese Entwicklungsumgebung vorgestellt, ehe mit der Erläuterung des zu realisierenden Szenarios sowie der Darstellung der wesentlichen Implementierungsschritte fortgefahren wird. Anschließend wird ein Blick auf die Laufzeit des Beispiels geworden, bevor ein Fazit

die wesentliche Erkenntnisse der Implementierung zusammenfasst. Darin wird auch die Rolle der modellgetriebenen Entwicklung Gegenstand der Betrachtungen sein. Bei der Umsetzung des Szenarios steht die lose Kopplung unter Verwendung der BPMN im Mittelpunkt, da sie im Zentrum der Grundarchitektur steht.

### *4.2.1 SAP NetWeaver Composition Environment*

Mit dem SAP NetWeaver Composition Environment (SAP NetWeaver CE) stellt die SAP eine Entwicklungs- und Laufzeitumgebung für die Entwicklung von SOA-basierten Anwendungen zur Verfügung. Die in diesem Produkt zusammengefassten Werkzeuge wurden ursprünglich unabhängig voneinander entwickelt und erst durch die Einführung von SAP NetWeaver CE zusammengeführt und aufeinander abgestimmt. Die Anfänge von CE reichen bis ins Jahr 2006 zurück, als die Umgebung auf der SAP eigenen Hausmesse SAP TechEd (Technical Education) einer breiten Öffentlichkeit vorgestellt wurde. Seitdem wurde das Composition Environment kontinuierlich weiterentwickelt und umfasst in Version 7.2 die in Abbildung 33 dargestellten Komponenten (in Anlehnung an Rauscher & Stiehl 2008).

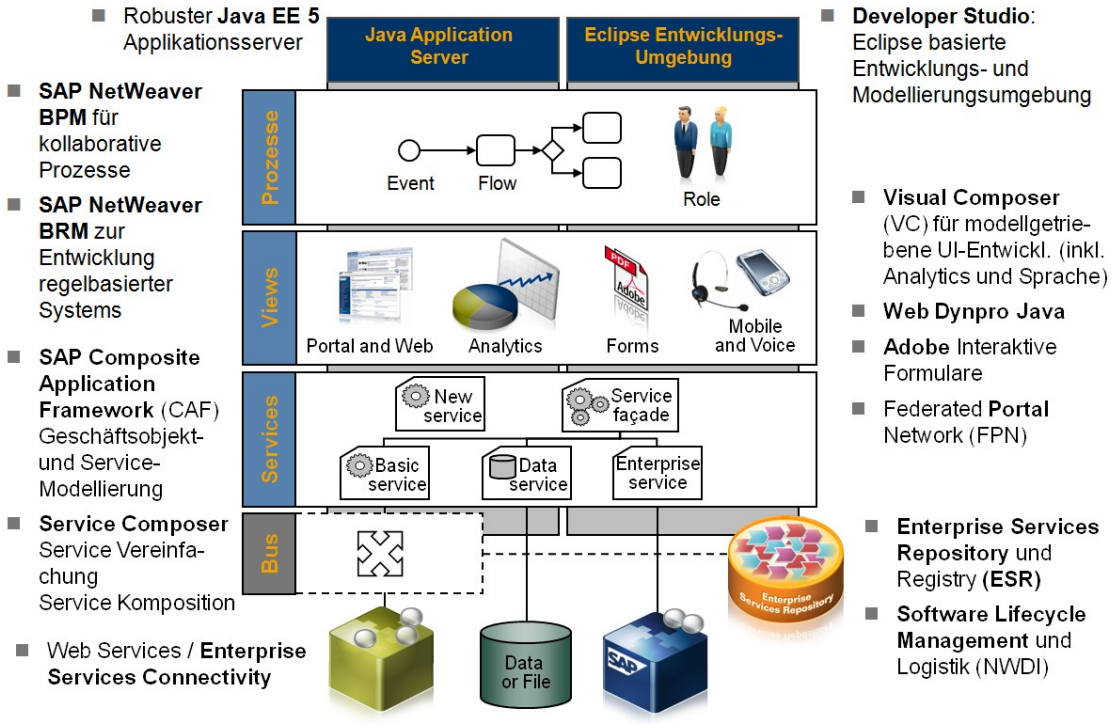



*Abbildung 33: Komponenten des SAP NetWeaver Composition Environments*

Das Fundament des gesamten Stacks ist ein Java EE 5-kompatibler und zertifizierter Applikationsserver. Er unterstützt die gängigen Standards wie EJB 3.0, JSF 1.2, JSP 2.1, JMS 1.1, JPA, SDO 2.1, JMX 1.2, JAX-WS 2.0 etc. Ebenfalls Bestandteil von SAP NetWeaver CE ist die Entwicklungs- und Modellierungsumgebung SAP NetWeaver Developer Studio (NWDS). Das SAP NetWeaver Developer Studio basiert auf dem Eclipse-Framework und erlaubt daher die Integration Eclipse-konformer Perspektiven. Das SAP NetWeaver Developer Studio ist wiederum eng mit der SAP NetWeaver Development Infrastructure (NWDI) verknüpft, die für das Software Lifecycle Management der Entwicklungskomponenten verantwortlich ist. Typische Aufgaben sind die Versionsverwaltung sowie der Transport aller zu einer Anwendung gehörenden Komponenten entlang der Entwicklungslandschaftskette (Entwicklung – Integration – Qualitätssicherung – Produktion).

Auffällig an der Architektur von SAP NetWeaver CE sind die verschiedenen Schichten, die denen von Verbundanwendungen entsprechen. Damit stehen für die Prozess-, Benutzeroberflächen- und Service- bzw. Objektschicht spezialisierte Werkzeuge zur Verfügung. Im Detail sind dies:

- In der Service-Schicht befindet sich das Composite Application Framework (CAF). CAF unterstützt sowohl die modellgetriebene Entwicklung von Geschäftsobjekten als auch die programmatische Implementierung von composite-spezifischer Geschäftslogik. Ergänzt wird CAF um den Service Composer, der mittels eines grafischen Editors die Modellierung einfacher Service-Verknüpfungen erlaubt. Dazu wird auf bereits existierende Dienste zurückgegriffen, die sich in der dem UDDI-Standard folgenden Registry befinden. Die Registry arbeitet dabei nicht nur mit dem Composite Application Framework und dem Service Composer eng zusammen, sondern steht auch den höheren Schichten zur Verfügung. Neben der Registry beinhaltet SAP NetWeaver CE auch ein Enterprise Services Repository, in dem Service-Schnittstellen verwaltet werden können und das bei Auslieferung bereits mit den Schnittstellen sämtlicher bei SAP entworfenen Dienste gefüllt ist. Sie dienen den Kunden als Vorbild für eigene Schnittstellenentwürfe. Das Repository deckt somit die SOA-Governance-Prozesse in den Unternehmen ab.

Zur Anbindung von Diensten aus dem Composite Application Framework bzw. Service Composer heraus stellt SAP NetWeaver CE eine einheitliche, auf Web Services basierende Infrastruktur zur Verfügung. CAF und Service Composer stellen ihrerseits die neu entwickelten Dienste wiederum als Web Services bereit, die anschließend von den nächsthöheren Schichten konsumiert werden können.

- In der UI- (oder auch Views-)Schicht verfügt das SAP NetWeaver Composition Environment über mehrere Technologien für die unterschiedlichsten Anwendungsfälle. So wird für die Entwicklung weniger komplexer Benutzeroberflächen der SAP NetWeaver Visual Composer (SAP NetWeaver VC) verwendet. Er erlaubt mit wenigen Klicks das Einbinden von Services, aus denen er über die Interpretation der Ein- und Ausgabeparameter Formulare und Tabellen automatisch generiert. Für den Entwickler reduziert sich die Arbeit auf die optische Anpassung der generierten Benutzeroberfläche und das Hinzufügen von Verifikationsskripten sowie Logik zur Steuerung des Bildschirmflusses. Dazu steht dem Entwickler ein grafischer Editor zur Verfügung. SAP NetWeaver VC wird zudem für die grafische Ausgabe von analytischen Auswertungen herangezogen. Insbesondere wenn Diagramme und Charts grafisch anspruchsvoll dargestellt werden sollen, bietet die Adobe-Flex-Integration vielfältige Möglichkeiten, die Benutzeroberfläche ansprechend zu gestalten. Schließlich wird der Visual Composer auch für die Modellierung von Sprachapplikationen herangezogen. Dabei werden im Visual Composer die Abläufe (Computer spricht, Warten auf Benutzerreaktion, Service-Aufruf) grafisch modelliert.

  Im Visual Composer kann nicht programmiert werden. Entwickler müssen folglich mit den von Haus aus gelieferten Funktionalitäten auskommen. Für anspruchsvollere Oberflächen empfiehlt sich daher der Einsatz von Web Dynpro. Dabei handelt es sich um ein von SAP auf Java Servlet-Basis entwickeltes UI-Framework mit einer umfangreichen Komponentenbibliothek, die neben aufwändigen UI-Komponenten wie Kalender, Navigationsbäumen, oder Tabellen mit integrierten Filter-, Sortier- und Personalisierungsfunktionen auch Dienste zur Verfügung stellt, die insbesondere im Geschäftsumfeld von Bedeutung

sind. Dazu gehören beispielsweise Wertehilfen, Internationalisierung, Scrollen in Massendaten oder der Zugriff für behinderte Menschen (Stichwort: Barrierefreiheit).

- SAP NetWeaver Visual Composer und Web Dynpro sind typische Online-Technologien. Für Anwendungsfälle, die offline abgewickelt werden müssen, stellt SAP alternativ interaktive Formularen zur Verfügung, die auf Adobe's PDF-Format basieren. So lädt beispielsweise ein Service-Techniker zu Beginn seines Arbeitstages seinen Arbeitsvorrat lokal auf sein Notebook und fährt anschließend zu seinen Kunden, um Reparaturen durchzuführen. Noch beim Kunden will er seine Arbeitsdaten erfassen: benötigte Zeit, benutztes Material etc. Am Abend möchte er dann die gesammelten Unterlagen an das Backend-System übertragen. Ein solches Szenario kann umgesetzt werden, indem sich der Techniker seine Arbeitsaufträge in Form von PDF-Dateien auf sein Notebook lädt. Diese kann er beim Kunden ausfüllen und wiederum lokal abspeichern, ohne dass dafür eine Online-Verbindung zur Firma bestehen müsste, was für Außendienstmitarbeitern ohnehin nicht vorausgesetzt werden kann. Er kann zudem Ausdrucke anfertigen und diese dem Kunden überlassen. Der Datentransfer zum Backend erfolgt schließlich durch einen Upload der PDF-Dateien, sobald er sich wieder ans Firmennetz angeschlossen hat.

- Die derart entwickelten Oberflächen können in ein leichtgewichtiges Portal integriert werden, das ebenfalls mit SAP NetWeaver CE ausgeliefert wird. Leichtgewichtig deshalb, weil das in CE integrierte Portal nicht alle Funktionalitäten des vollständigen SAP-Portal-Produktes enthält. So wurden die KMC-Funktionalitäten (Knowledge Management und Collaboration) nicht mitaufgenommen. Neben der eigenständigen Ablauffähigkeit des CE-Portals bietet es auch eine enge Anbindung an das vollständige SAP-Portal-Produkt, indem die auf SAP NetWeaver Composition Environment entwickelten Benutzeroberflächen dort leicht zu integrieren sind. Dies wird als Federated Portal Network (FPN) bezeichnet, bei dem das CE-Portal die Rolle eines Inhaltsproduzenten übernimmt.

- In der obersten Schicht schließlich sorgt SAP NetWeaver Business Process Management (SAP NetWeaver BPM) mit seinem grafischen BPMN-Editor für die Verbindung von Benutzeroberflächen und Service-Aufrufen zu kollaborativen Geschäftsprozessen. Dabei wird im Gegensatz zu anderen Herstellern der BPMN-Prozess ohne Transformation nach BPEL direkt zur Ausführung gebracht. Die BPMN-Prozess-Engine überwacht den Prozessablauf, informiert Prozessteilnehmer mithilfe entsprechender Meldungen über anstehende Aufgaben und gibt Administratoren über dedizierte Werkzeuge die Möglichkeit, laufende Prozesse zu beobachten, Aufgaben anderen Mitarbeitern zuzuordnen oder Prozesse zu beenden. Sie bietet die wesentlichen Funktionalitäten, die man zur Erstellung leichtgewichtiger personenbezogener Prozesse benötigt.
  Ergänzt wird SAP NetWeaver BPM um eine Entwicklungs- und Laufzeitumgebung für regelbasierte Anwendungen. SAP NetWeaver Business Rules Management (SAP NetWeaver BRM) deckt dabei sowohl Rete-basierte als auch sequenziell auszuführende Regelmengen ab. Insbesondere die Kombination von Prozessen und Regeln erlaubt den Entwurf sehr flexibler und dabei gleichzeitig stabiler Architekturen, von denen auch Verbundanwendungen profitieren können, wie in einem separaten Kapitel (Kapitel 5.7 - Flexibilitätsgewinn durch die Verwendung von Regelwerken in Kombination mit analytischen Anwendungen) noch zu sehen sein wird.

Mit dem SAP NetWeaver Composition Environment steht somit ein Produkt zur Verfügung, dass die Entwicklung von Verbundanwendungen optimal unterstützt. Bei der Entwicklung der Werkzeuge wurde auf die bestmögliche Unterstützung des Anwendungsentwicklers geachtet, um dessen Produktivität zu steigern. Folglich finden sich in allen Werkzeugen modellgetriebene Ansätze wieder, die je nach Bedarf durch Java-Code an den Stellen ergänzt werden können, wo eine Modellierung nicht mehr möglich ist. Durch diese Kombination können sowohl Produktivitätssteigerung als auch Flexibilität erreicht werden.
Aufgrund der speziellen Ausrichtung des Composition Environments auf die Unterstützung der Entwicklung von Verbundanwendungen, war das Produkt für die Implementierung der Composite-Grundarchitektur prädestiniert. Nach einer kurzen Be-

schreibung des umzusetzenden Szenarios werden anschließend die wesentlichen Entwicklungsschritte im SAP NetWeaver Composition Environment skizziert.

### 4.2.2 Implementierungsszenario: vereinfachter Bestellprozess

In Anlehnung an den Beispielprozess aus Kapitel 3.3.2.3, wurde die Abwicklung einer Bestellung auch für die Implementierung herangezogen. Der für die Umsetzung vereinfachte Prozessablauf ist in Abbildung 34 dargestellt.

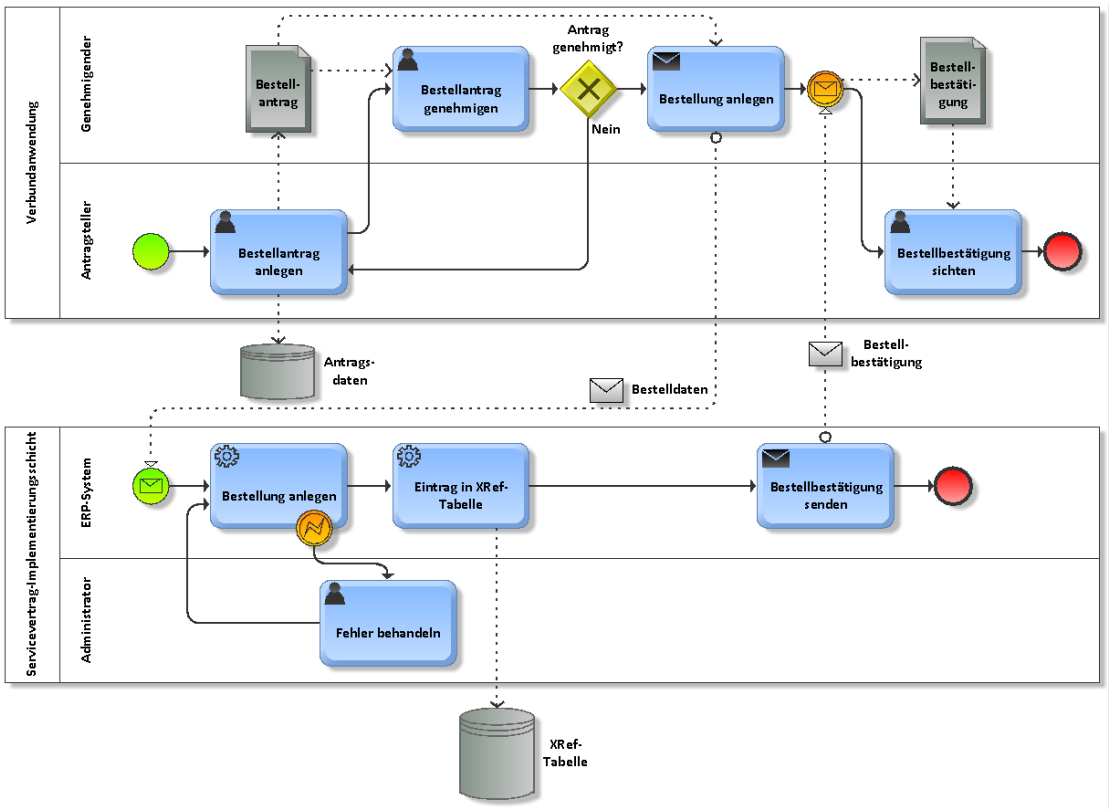



***Abbildung 34: Vereinfachter Bestellprozess in BPMN***

Der Prozess beginnt mit der Eingabe des Bestellantrags durch den Antragsteller. Da asynchrone Aufrufe an die Servicevertrag-Implementierungsschicht aus Benutzeroberflächen heraus vermieden werden sollen, werden die Daten nach Beendigung der Eingabe lokal in der Verbundanwendung in dem Datenspeicher *Antragsdaten* abgelegt. Dem Antragsteller wird eine automatisch generierte Antragsnummer angezeigt. Der Antrag wird anschließend an den Genehmigenden zur Überprüfung weitergeleitet. Wird er abgelehnt, hat der Antragsteller nochmals die Möglichkeit, seine Eingaben zu korrigieren und erneut zur Genehmigung vorzulegen. Wird der Antrag schließlich an-

genommen, erfolgt durch Aufruf der sendenden Aufgabe *Bestellung anlegen* die Übergabe an die Servicevertrag-Implementierungsschicht. Der Prozess der Verbundanwendung unterbricht in der Zwischenzeit die weitere Verarbeitung an dem empfangenden Nachrichten-Zwischenereignis.

Die Implementierungsschicht übergibt die Bestellung an das dafür zuständige Backend-System. Tritt dabei ein Fehler auf, wird zeitnah der für das System zuständige Administrator informiert. Nach dessen Eingreifen wird der Serviceaufruf wiederholt. Fortgesetzt wird der Prozess bei erfolgreicher Übergabe an das ERP-System mit der Anlage eines Eintrags in der Cross-Referenz-Tabelle, die bekanntlich die in der Verbundanwendung erzeugte Antragsnummer mit der im Backend generierten Auftragsnummer in Relation setzt. Als letzte Aktion verschickt der technische Prozess die Bestellbestätigung an den wartenden Prozess der Verbundanwendung und beendet sich anschließend. Die empfangenen Daten der Bestellbestätigung werden nun wiederum dem Antragsteller zur Sichtung vorgelegt. Nach dessen Bestätigung endet auch der fachliche Prozess.

### *4.2.3 Grundlegende Entwicklungsschritte*

Im Rahmen dieser Arbeit wird kurz auf die wesentlichen Entwicklungsschritte zur Umsetzung des beschriebenen Szenarios mit Hilfe des SAP NetWeaver Composition Environments eingegangen. Für eine detaillierter Erläuterung sei auf Stiehl 2010c verwiesen.

#### 4.2.3.1 Geschäftsprozess und technischer Prozess

Gemäß des empfohlenen Top-Down-Vorgehens steht der Geschäftsprozess so wie in Abbildung 34 dargestellt am Beginn der Implementierung. SAP NetWeaver BPM stellt zur BPMN-Modellierung ein Eclipse-Plugin namens Process Composer zur Verfügung. Er erlaubt die grafische Modellierung des Prozesses. Das Ergebnis ist in Abbildung 35 dargestellt.

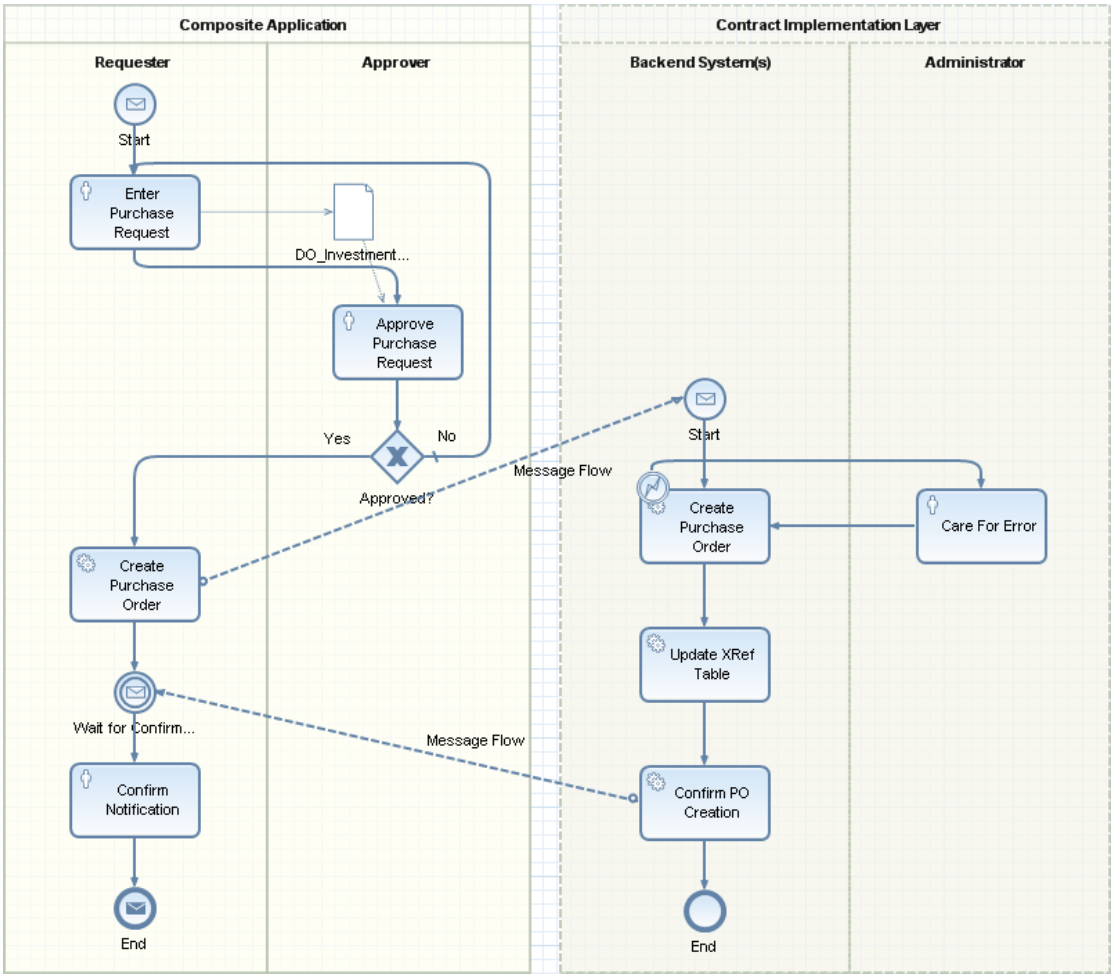



***Abbildung 35: Vereinfachter Bestellprozess modelliert im Process Composer***

Im Vergleich zur Vorgabe aus Abbildung 34 fallen einige Unterschiede auf. So unterstützt die derzeitige Version von SAP NetWeaver BPM keine sendenden Aufgaben, um explizit asynchrone Aktionen darzustellen. SAP NetWeaver BPM erlaubt zwar den asynchronen Aufruf von Services, allerdings werden sowohl synchrone als auch asynchrone Aufrufe durch die Service-Aufgabe ausgeführt. Von daher wurde sowohl für den *Create Purchase Order*- als auch für den *Confirm PO Creation*-Aufruf die Service-Aufgabe verwendet.

Auch die Datenbehandlung unterscheidet sich leicht von der ursprünglichen Planung. Dies hängt mit der Umsetzung von Datenobjekten in SAP NetWeaver BPM zusammen: die Datenobjekte repräsentieren im Prozessmodell den Prozesskontext, also die Daten, die ausschließlich während der Prozessausführung den einzelnen Prozessschritten zur Verfügung stehen und folglich nach Prozessbeendigung nicht mehr zugreifbar sind. Allein die Bereitstellung des Datenobjekts im Modell genügt, um später

den gesamten Prozessfluss ausmodellieren zu können. Dabei übernehmen die dargestellten Assoziationen vom *Enter Purchase Request*-Schritt zum Datenobjekt bzw. vom Datenobjekt zur *Approve Purchase Request*-Aufgabe keinerlei semantische Funktionen. Sie dienen ausschließlich zur Dokumentation.

Dasselbe gilt für die Nachrichtenflüsse zwischen den beiden Pools. Auch sie übernehmen Dokumentationsaufgaben. Überhaupt ist aus Abbildung 35 lediglich der *Composite Application*-Pool ausführbar, da in SAP NetWeaver BPM immer nur ein Prozess eines Prozessmodells ausführbar sein kann. Alle anderen Pools helfen, Zusammenhänge darzustellen, ohne dabei Einfluss auf die Laufzeit zu haben. Sie sind im Prozessmodell an der gestrichelten Pool-Umrandung zu erkennen. Folglich musste auch für die Servicevertrag-Implementierungsschritt ein zweites Prozessmodell angelegt werden. Es ist in Abbildung 36 dargestellt. Diesmal wurde bewusst auf die Darstellung des Zusammenspiels mit der Composite Application verzichtet, da das Gesamtszenario bereits aus dem Modell der Abbildung 35 hervorgeht.

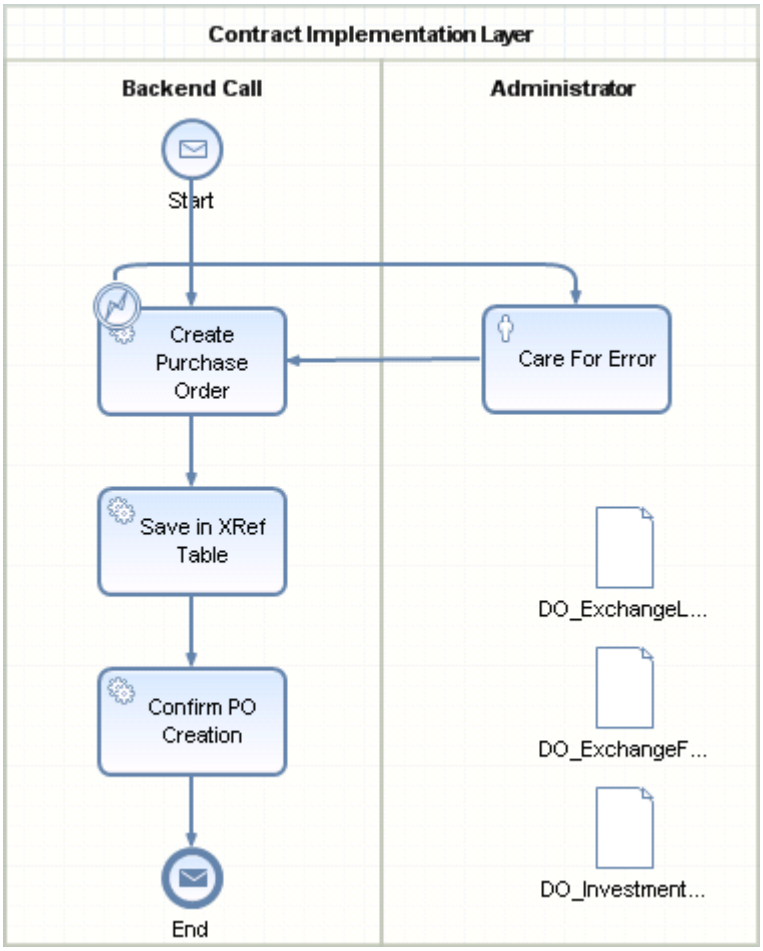



***Abbildung 36: Servicevertrag-Implementierungsschicht implementiert im Process Composer***

Fortgesetzt wird die Implementierung mit der Bestimmung des Datenmodells, auf dem das Gesamtszenario operiert. Gemäß des Prozessmodells beginnt der Ablauf mit der Anlage der Bestellanforderung. Dazu wird der Antragsteller nach Produkten suchen wollen, die sich aus einer Produktnummer, einer Produktbezeichnung, sowie einem Preis samt Währung zusammensetzen. Nach Auswahl eines Produktes wird er die Bestellmenge ergänzen und die Bestellung ggf. mit einem Kommentar ergänzen wollen. Wird das ausgewählte Produkt und die Bestellmenge bestätigt, werden gemäß der Implementierungsrichtlinie für Verbundanwendungen die Daten zunächst lokal gespeichert, woraus sich eine interne Reservierungsnummer ergibt.

Die Bestelldaten erscheinen daraufhin beim Genehmigenden. Zur Erleichterung seiner Entscheidung werden ihm neben den Eingabedaten auch der Gesamtpreis angezeigt. Die eigentliche Genehmigung oder Ablehnung des Antrags wird über eine pas-

sende Checkbox ermöglicht. Auch er kann seine Entscheidung durch einen Kommentar näher erläutern.

Nach erfolgter Genehmigung wird die interne Reservierungsnummer zusammen mit der Produktnummer und der Bestellmenge an die Servicevertrag-Implementierungsschicht übergeben. Diese wiederum übernimmt den Aufruf des zuständigen Backend-Systems, woraus sich eine externe Auftragsnummer ergibt, die zusammen mit der internen Nummer in der Cross-Referenztabelle abgelegt wird. Die erzeugte externe Nummer wird als Beweis der erfolgreichen Verbuchung an den Prozess der Verbundanwendung zurückgeliefert und dem Antragsteller in seiner Bestellbestätigung angezeigt. Hierbei handelt es sich natürlich eindeutig um einen Verstoß gegen die Composite-Implementierungsregeln, da externe Nummern nicht in einer Verbundanwendung auftreten dürfen. Zur sofortigen Verifikation der erfolgreichen Backend-Integration wurde für die Implementierung des Beispielszenarios allerdings bewusst auf die strenge Richtlinienauslegung verzichtet.

Zudem wird zur Vereinfachung des Szenarios von lediglich einem anzubindenden Backend-System ausgegangen. Die prinzipielle Implementierungsidee lässt sich allerdings leicht auf mehrere Systeme erweitern.

#### 4.2.3.2 Daten und Datentypen

Die obige Beschreibung lässt nun Rückschlüsse auf die einzelnen Datenfelder zu. Bleibt die Frage des zu verwendenden Datentypsystems. Hier stehen dem Entwickler mehrere Optionen offen: Java-Datentypen, XSD-Datentypen, Datentypen anerkannter Standards, so wie sie in Kapitel 3.2.7 diskutiert wurden. Für die Implementierung dieses Beispiels wurden XSD-Datentypen gewählt, da sie über einen generischeren Charakter als Java-Datentypen verfügen und dadurch für heterogene Umgebungen besser geeignet sind. Details zu XSD-Datentypen finden sich in der Spezifikation des W3C zu XML-Schema (W3C 2004). In Summe können die benötigten Felder einschließlich Datentypen in XSD wie folgt dargestellt werden:

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<schema
 xmlns=http://www.w3.org/2001/XMLSchema
 xmlns:tns=http://www.example.org/approvalprocess
 elementFormDefault="qualified"
 targetNamespace="http://www.example.org/approvalprocess">
```

```
    <elementname="InvestmentApprovalProcess" type="tns:InvestmentApprovalProcess"/>
    <complexType name="InvestmentApprovalProcess">
      <sequence>
        <element name="region" type="string"/>
        <element name="productID" type="string"/>
        <element name="quantity" type="decimal"/>
        <element name="price" type="decimal"/>
        <element name="totalAmount" type="decimal"/>
        <element name="comment" type="string"/>
        <element name="internalReservationID" type="string"/>
        <element name="externalOrderID" type="string"/>
        <element name="approved" type="boolean"/>
      </sequence>
    </complexType>
</schema>
```

Neben den bereits erwähnten Daten fällt das Element *region* in der Schemadefinition auf. Das Feld enthält die Region, aus der die Bestellanfrage stammt. Dieses Feld wird erst in einer späteren Prozesserweiterung eine Rolle spielen, wenn die Entscheidung über eine Genehmigung durch ein Regelwerk basierend auf der Gesamtsumme und der Region getroffen wird.

Die verwendeten Datentypen sind selbsterklärend. Erwähnenswert ist allenfalls die Wahl des Typs `string` für die Identifier *productID*, *externalOrderID* und *internalReservationID*. Bei den internen Nummern hätte der Entwickler die Entscheidung noch selbst in der Hand, von welchem Datentyp das Feld sein sollte. Bei externen Nummern ist dies schon nicht mehr der Fall. Da binäre Darstellungen die Nachvollziehbarkeit u.a. auch bei der Kommunikation zwischen den Szenariobeteiligten nur unnötig erschweren würde, z.B. bei der manuellen oder automatisierten Suche nach Nachrichten, liegt eine leicht lesbare Darstellung nahe, die durch Verwendung von `string` gewährleistet wird. Entsprechend wird `string` aus denselben Gründen auch für die internen Nummern verwendet.

#### 4.2.3.3 Serviceverträge

Zur Definition der Serviceverträge zwischen Verbundanwendung und der dazugehörigen Implementierungsschicht sind die betriebswirtschaftlichen Anforderungen der Composite zu betrachten. In Summe lassen sich drei Verträge identifizieren.

- Eine synchrone Schnittstelle zur Suche nach Produkten. Sie wird aus der Benutzeroberfläche der Benutzeraufgabe zur Anlage der Bestellanforderung benö-

tigt. Zur Vereinfachung sei angenommen, dass der Antragsteller nach Produktnummern bzw. nach Produktbeschreibungen suchen möchte. Als Ergebnis erwartet er eine Liste von Produkten, die den Suchkriterien entsprechen und neben Produktnummer und -beschreibung auch Informationen über Preis und Währung beinhalten. In XSD lässt sich sowohl die Suche als auch die Ergebnisliste wie folgt darstellen:

```
<xsd:complexType name="Request">
    <xsd:sequence>
           <xsd:element name="productID" type="xsd:string"/>
           <xsd:element name="description" type="xsd:string"/>
     </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="ProductEntry">
    <xsd:sequence>
           <xsd:element name="productID" type="xsd:string"/>
           <xsd:element name="description" type="xsd:string"/>
           <xsd:element name="price" type="xsd:decimal"/>
           <xsd:element name="currency" type="xsd:string"/>
    </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="Response">
    <xsd:sequence>
           <xsd:element maxOccurs="unbounded" minOccurs="0"
           name="ProductList" type="ProductEntry"/>
    </xsd:sequence>
</xsd:complexType>
```

- Zur asynchronen Instanziierung des technischen Prozesses zur Speicherung der Bestelldaten werden lediglich das bestellte Produkt, die Bestellmenge und die interne Reservierungsnummer übergeben. Als XSD-Darstellung ergibt sich daraus:

```
<xsd:element name="createPOOperation">
    <xsd:complexType>
           <xsd:sequence>
                  <xsd:element name="internalReservationID" type="xsd:string" />
                  <xsd:element name="productID" type="xsd:string" />
                  <xsd:element name="quantity" type="xsd:decimal" />
           </xsd:sequence>
    </xsd:complexType>
```

```
</xsd:element>
```

- Schließlich wird zur Bestätigung der erfolgreichen Bestelldatenverbuchung eine asynchrone Schnittstelle benötigt, die als Inhalt lediglich die interne Reservierungsnummer und die im Backend erzeugte externe Bestellnummer umfasst. Die interne Reservierungsnummer wird zur Korrelation mit dem Prozess benötigt, der den Schreibprozess initiiert hat. In XSD stellt sich diese Schnittstelle wie folgt dar:

```
<xsd:element name="confirmPOCreationOperation">
    <xsd:complexType>
        <xsd:sequence>
            <xsd:element name="internalReservationID" type="xsd:string"/>
            <xsd:element name="externalOrderID" type="xsd:string"/>
        </xsd:sequence>
    </xsd:complexType>
</xsd:element>
```

Damit ist die Außendarstellung der Verbundanwendung abgeschlossen. Werden die Schnittstellen zudem in einem Repository geführt, wie dies in dieser Arbeit bereits empfohlen wurde, so ist für die Entwickler leicht zu ermitteln, welche betriebswirtschaftlichen Funktionalitäten für die Verbundanwendung erbracht werden müssen, damit diese korrekt funktionieren kann. Zudem lassen sich aus den Definitionen mit Werkzeugen neuester Generation Proxies generieren, leere Codefragment, die vom Entwickler nur noch mit der benötigten Geschäftslogik gefüllt werden muss.

Die Schnittstellen selbst lassen sich leicht mit gängigen XSD-Editoren erstellen. Das SAP NetWeaver Composition Environment bedient sich dazu des XSD-Editors, der Bestandteil des Eclipse WTP-Projektes (Web Tools Platform) ist.

#### 4.2.3.4 Persistenz

Eine wesentliche Idee der Umsetzung der losen Kopplung zwischen Verbundanwendung und Servicevertrag-Implementierungsschicht ist die lokale Zwischenspeicherung der Antragsdaten als Teil der Verbundanwendung, damit der Endanwender direkt in der Oberfläche zur Eingabe der Antragsdaten eine Bestätigung der erfolgreichen Abspeicherung erhält. Dazu bedarf es einer lokalen Persistenz. In der Java-Welt hat sich dafür das Java Persistence API (JPA) bewährt. SAP hat für das SAP NetWea-

ver Composition Environment basierend auf JPA eine zusätzliche Schicht entwickelt, die dem Entwickler die grafische Modellierung von Objekten ermöglicht. Ausgehend von diesen Objektmodellen werden neben der automatischen Generierung von Datenbanktabellen auch klassische CRUD-Lebenszyklusmethoden (Create, Read, Update, Delete) sowie Suchmethoden erzeugt, die wiederum als Web Services exponiert und derart in Benutzeroberflächen und Prozesse integriert werden können. Das Modell für die interne Reservierung ist Abbildung 37 dargestellt.

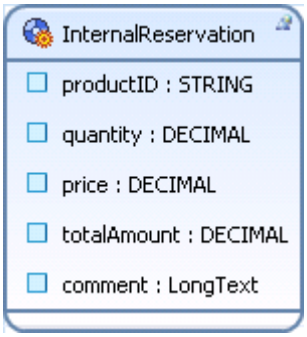



***Abbildung 37: Datenmodell einer Reservierung***

Es besteht aus dem gewählten Produkt (*productID*), der Bestellmenge (*quantity*), dem Preis (*price*), sowie der Gesamtsumme der Bestellung (*totalAmount*) und einem Kommentar des Antragstellers (*comment*). Die automatisch erzeugten Methoden zeigt Abbildung 38:

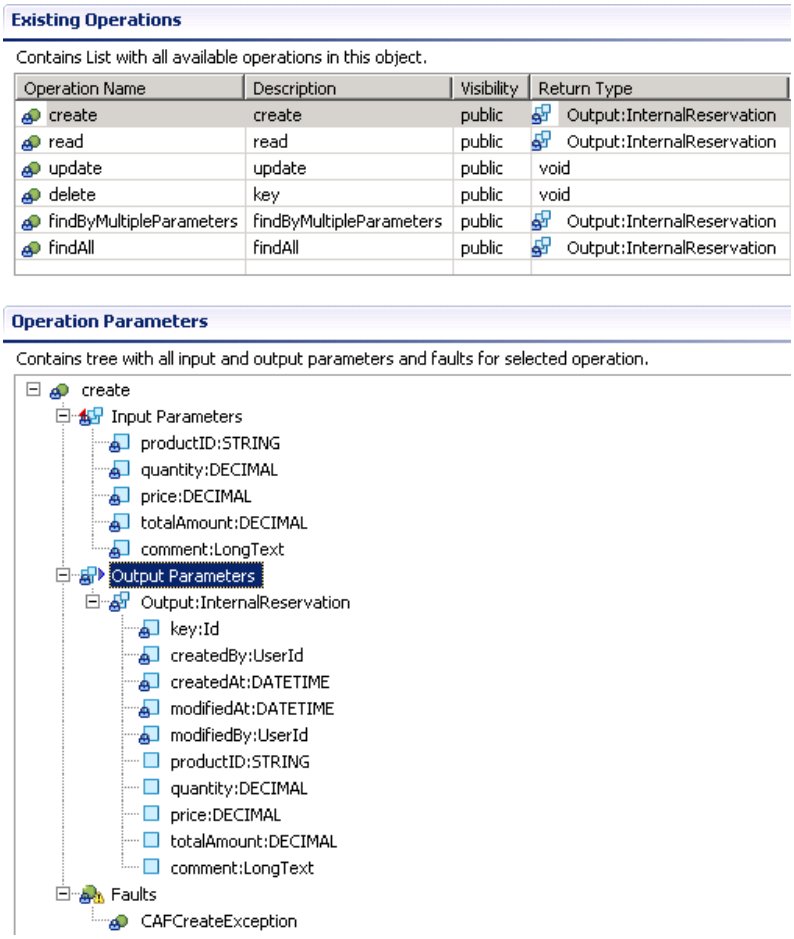



***Abbildung 38: Automatisch generierte Methoden der Reservierung***

In der Abbildung ist beispielhaft die *create*-Methode selektiert. Offensichtlich gibt der Methodenaufruf das vollständig erzeugte Objekt zurück. Dabei wird der eindeutige Primärschlüssel (Feld *key*) als auch die die Verwaltungsfelder (*createdBy, createdAt, modifiedBy* und *modifiedAt*) automatisch vom Framework erzeugt. Der Entwickler kann sich dadurch ausschließlich auf die Modellierung der betriebswirtschaftlich relevanten Felder konzentrieren.

Auch in der Servicevertrag-Implementierungsschicht wird auf die lokale Persistenz zurückgegriffen: in der Cross-Referenztabelle werden die interne Reservierungsnummer der Verbundanwendung mit der extern erzeugten Auftragsnummer des Backend-Systems in Beziehung gesetzt. Das dazugehörige Objektmodell mit den beiden Feldern *internalReservationID* und *orderID* ist in Abbildung 39 dargestellt.

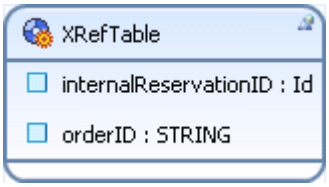



***Abbildung 39: Datenmodell der Cross-Referenztabelle***

Im Unterschied zum *InternalReservation*-Objekt wird die *create*-Methode der *XRefTable* diesmal direkt aus dem Prozessmodell heraus aufgerufen. Lediglich die technische Implementierung über Web Services ist identisch.

#### 4.2.3.5 Benutzeroberflächen

Wie den Prozessmodellen zu dem Beispiel zu entnehmen ist, werden insgesamt vier Benutzeroberflächen zur Implementierung des Szenarios verlangt, wobei insbesondere die UIs *Enter Purchase Request* sowie *Approve Purchase Request* im Rahmen dieser Arbeit von besonderem Interesse sind, da mit ihnen das Zusammenspiel zwischen Antragsteller und Genehmigenden umgesetzt wird. Die beiden Oberflächen *Care For Error* und *Confirm Notification* vervollständigen zwar das Prozessmodell, liefern hingegen keine neuen Erkenntnisse.

Die Erzeugung von Benutzeroberflächen gehört sicherlich zu den aufwändigsten Aufgaben bei der Erstellung von Anwendungen. Erschwerend kommt bei der Entwicklung von Verbundanwendungen hinzu, dass die Oberflächen mit dem Prozessframework zusammenarbeiten müssen: so muss das UI zur Anlage des Bestellantrags nicht nur die Beendigung der Eingabe in Form eines Ereignisses mitteilen. Es muss zudem für die Weitergabe der Daten an den Prozesskontext sorgen. Der Process Composer von SAP NetWeaver BPM unterstützt Entwickler dahingehend, als dass die Oberflächen aus den Daten des Prozesskontextes heraus generiert werden können. Bei der Generierung können aufgrund der Datendefinitionen im Prozesskontext automatisch Masken mit Eingabefeldern und Tabellen erzeugt werden. Die derart generierten Formulare müssen durch den Entwickler zwar noch nachbearbeitet werden, dennoch wird eine nicht unerhebliche Produktivitätssteigerung durch dieses Vorgehen erreicht. Für die Beispielimplementierung wurde allerdings bewusst auf eine Nachbearbeitung verzichtet, da primär die lose Kopplung zwischen fachlichen und technischen Prozess verifiziert werden soll.

Für den Prozessschritt *Enter Purchase Request* wurden zwei Versionen erstellt: eine einfache Lösung ohne Produkt-Suchmaske und eine aufwändigere Lösung mit Produkt-Suche. In beiden Fällen ist für die Implementierung eines lose gekoppelten Szenarios allerdings weniger die Suche nach Produkten von Bedeutung, als der Aufruf der lokalen Speicherung der Antragsdaten, damit der Benutzer schnellstmöglich ein visuelles Feedback zu seiner Datenspeicherung bekommt. Von daher ist in Abbildung 40 zunächst der Datenfluss der einfacheren Variante im SAP Visual Composer dargestellt.

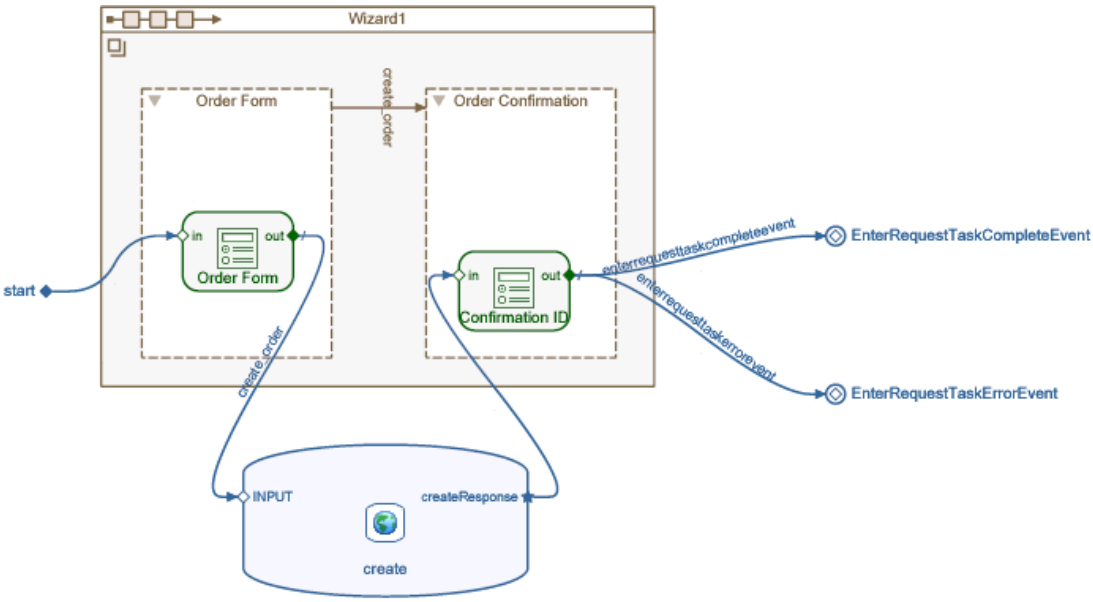



***Abbildung 40: Oberflächenentwicklung mit SAP Visual Composer***

In dem Modell ist ein Wizard-UI dargestellt, das aus den zwei Formularen *Order Form* (siehe Abbildung 41) und *Order Confirmation* (siehe Abbildung 42) besteht.

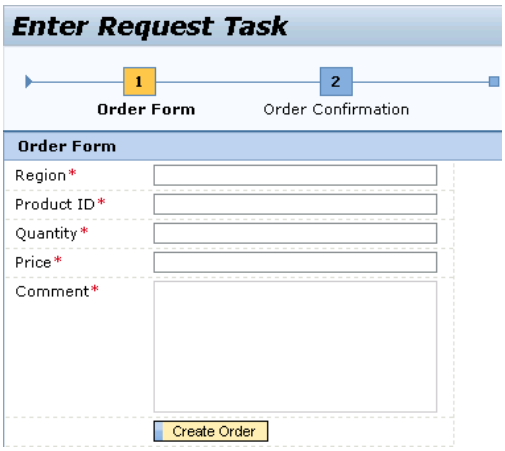



*Abbildung 41: Bestellformular*

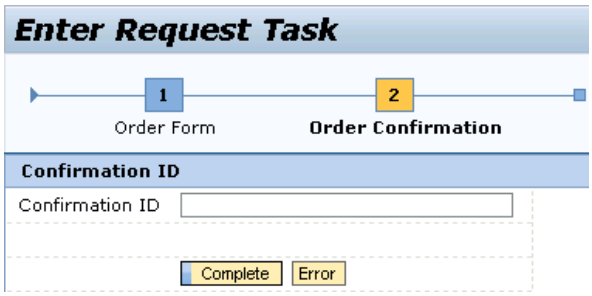



*Abbildung 42: Bestellbestätigung*

In dem *Order Form* Formular gibt der Anwender seine Bestelldaten ein. Hat er seine Eingaben beendet, klickt er auf den *Create Order* Button und die Daten werden per Web Service an die *create*-Methode des *InternalReservation*-Objekts übergeben. Dadurch wird letztendlich die lokale Persistierung der Daten erreicht. Die dabei erzeugte Reservierungsnummer wird im zweiten Schritt des Wizard-UIs dem Benutzer angezeigt. Dabei wird das Primärschlüsselfeld *key* im Visual Composer dem *Confirmation ID*-Feld der Maske zugeordnet (gemappt). Durch Klick auf *Complete* wird schließlich die Maske abgeschlossen und das Prozessframework über das Ende der Dateneingabe informiert. Gleichzeitig werden die eingegebenen Daten an den Prozesskontext übergeben. Die Benutzeroberfläche einschließlich einer Suchmaske ist in Abbildung 43 dargestellt. Für dieses UI wurde zusätzlich die Implementierung des Suchinterfaces aus Abschnitt 4.2.3.3 vorgenommen.

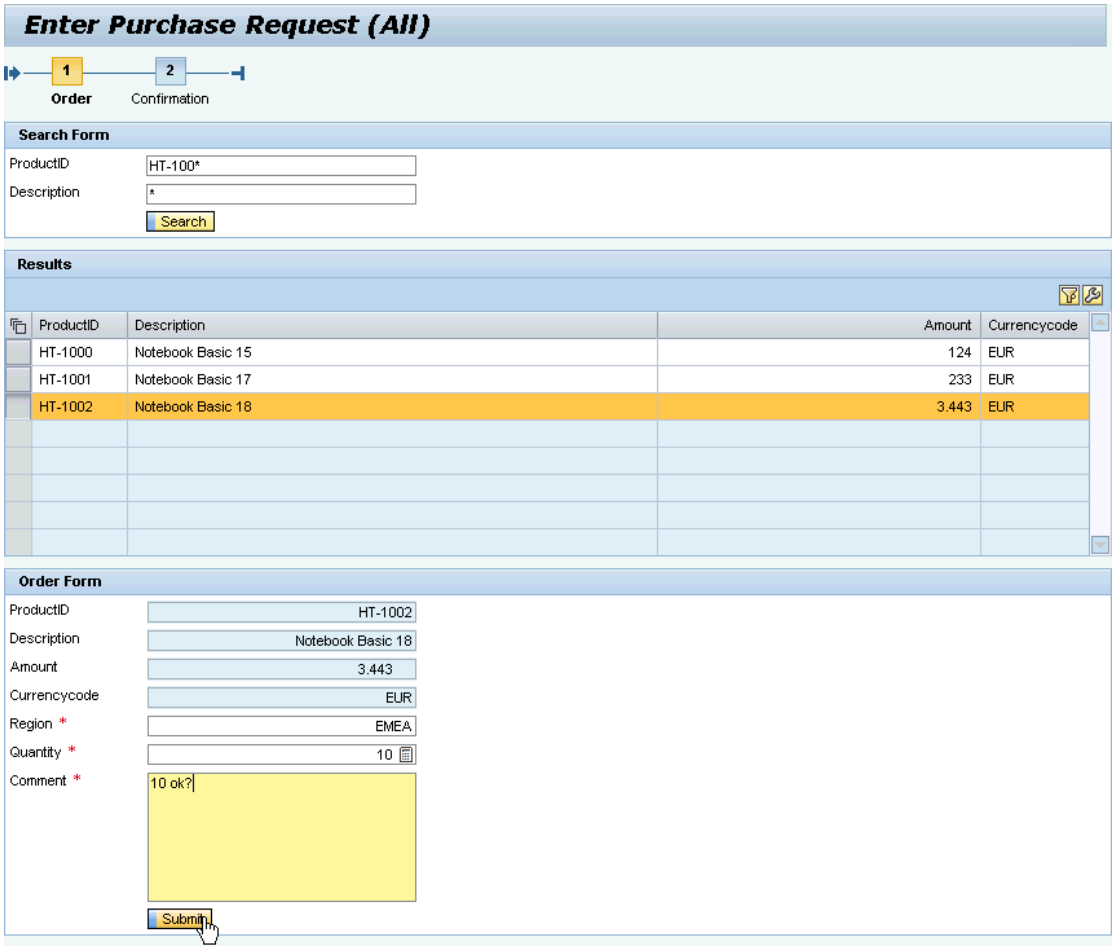



***Abbildung 43: Benutzeroberfläche zum Anlegen der Bestellung einschließlich Produktsuche***

Die Benutzeroberfläche zur Genehmigung der Bestellung wurde mit der Web Dynpro-Technologie implementiert. Im Gegensatz zum vorangegangenen Bestellformular sind aus dem UI heraus keine Service-Aufrufe notwendig. Es werden lediglich die vom Antragsteller eingegebenen Werte angezeigt. Der Genehmigende kann durch setzen einer entsprechenden Checkbox (*Approved*) den Antrag genehmigen oder ablehnen (Abbildung 44).

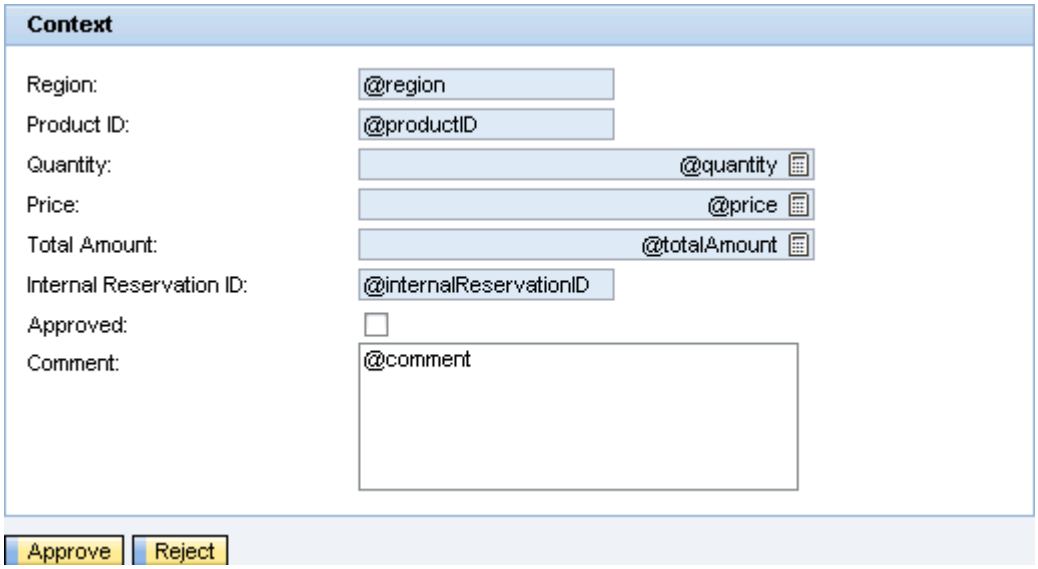



***Abbildung 44: Benutzeroberfläche zur Genehmigung des Antrags***

Auch diese Oberfläche ließ sich automatisch aus dem Prozesskontext heraus generieren. Datenübergabe und Zusammenspiel mit dem Prozessframework SAP NetWeaver BPM werden dadurch gewährleistet. Über den Zustand der *Approved*-Checkbox wird das Verhalten des nachfolgenden Gateways gesteuert und über das Kommentarfeld schließlich wird die Kommunikation zwischen den Prozessbeteiligten abgewickelt.

Bei den einzelnen Schritten stellt sich zwangsläufig die Frage, wer konkret diese Aufgaben ausführen wird. Den Schritten bereits zur Entwicklungszeit konkrete Benutzer zuzuordnen wäre allerdings zu starr. Von daher bedient sich SAP NetWeaver BPM eines Rollenkonzepts. Jeder Lane im Pool kann eine Prozessrolle zugeordnet werden, die anschließend mit konkreten Gruppen bzw. Rollen einer bereits im Unternehmen existierenden Benutzerverwaltung (z.B. ein LDAP-Verzeichnis, eine SAP-Benutzerverwaltung oder eine JDBC-Datenbank) verknüpft werden kann. Jede Aktivität innerhalb dieser Lane wird dadurch automatisch von dem/den Anwender/n ausgeführt, die den zugeordneten Gruppen/Rollen der Benutzerverwaltung angehören. Auf diese Weise wird erst zur Laufzeit ermittelt, welcher Anwender (bzw. welche Anwender) die Aufgabe letztendlich bearbeiten können. Zudem erhöht es die Flexibilität bei organisatorischen Änderungen, da dann automatisch die richtigen Bearbeiter aus der aktualisierten Benutzerverwaltung gezogen werden.

#### 4.2.3.6 Ergänzende Implementierungsdetails

Soweit wurden die wesentlichen Entwicklungsschritte für die Implementierung der einzelnen Bestandteile der Verbundanwendung erläutert. Die Benutzeroberflächen können den Benutzeraufgaben und die erzeugten Web Services den Service-Aufgaben des Prozessmodells zugeordnet werden. Für die Service-Aufgabe *Create Purchase Order* der Servicevertrag-Implementierungsschicht wird wiederum auf einen vom Backend-System bereitzustellenden Web Service zur Verbuchung der Auftragsdaten zurückgegriffen. Die Aufrufreihenfolge der Oberflächen und Services wird durch den modellierten Prozessfluss festgelegt. Der Prozess verknüpft die einzelnen Bestandteile folglich aufgrund der modellierten Prozesslogik.

Der bereits des öfteren angesprochene Datenaustausch zwischen den Prozessschritten und dem Prozesskontext wird durch Mapping realisiert. Dabei wird stets der Datenfluss aus einer Aktivität zum Prozesskontext bzw. vom Prozesskontext zur Aktivität modelliert, niemals jedoch der direkte Datenaustausch zwischen zwei Aktivitäten. Dieses Vorgehen gewährleistet die Erweiterung des Prozesses um weitere Schritte, ohne dass dadurch der bereits modellierte Datenfluss unterbrochen würde. Die Mappings sind dabei den einzelnen Aktivitäten zugeordnet. Es wird zwischen einem Input- (vom Prozesskontext zur Aktivität) und einem Output-Mapping (von der Aktivität zum Prozesskontext) unterschieden. Die Abbildungen 45 und 46 zeigen die jeweiligen Mappings für den *Approve Purchase Request*-Schritt.

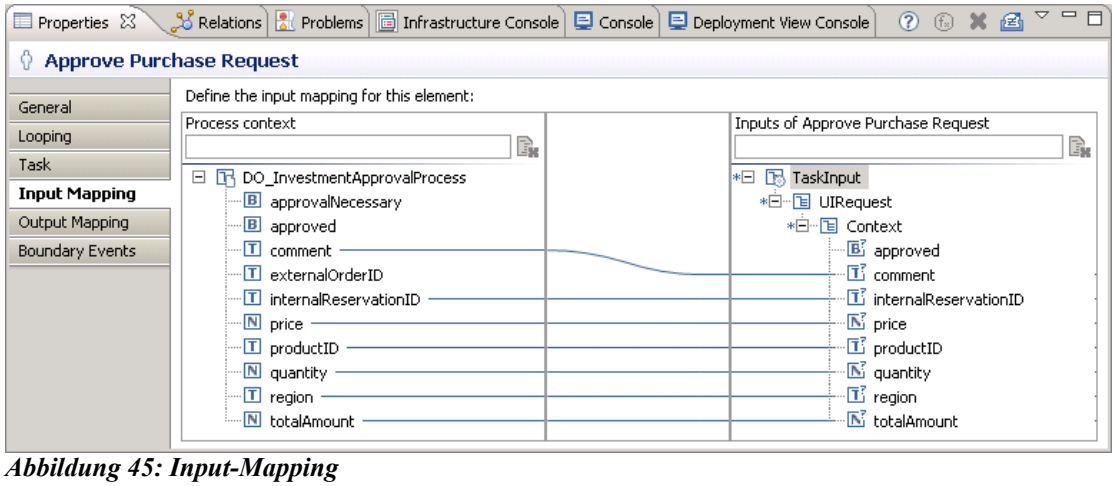



***Abbildung 45: Input-Mapping***

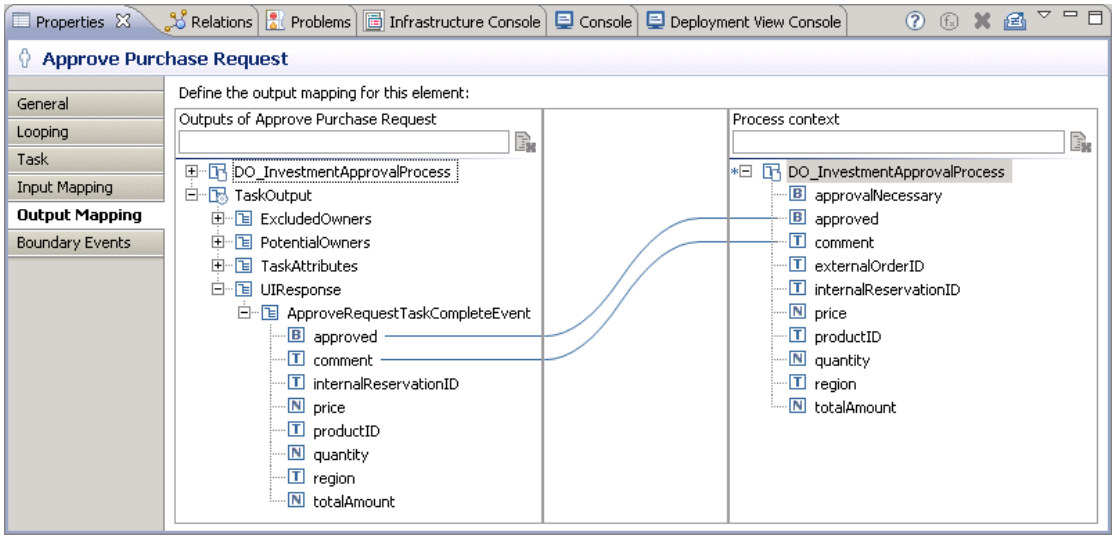



***Abbildung 46: Output-Mapping***

Der Datenfluss erfolgt bei den Mappings jeweils von links nach rechts. Im Output-Mapping wurden nur die Felder gemappt, die in der dazugehörigen Benutzeroberfläche auch verändert werden können. Wie dies dem Screenshot aus Abbildung 46 zu entnehmen ist, sind dies lediglich die Approval-Checkbox und das Kommentarfeld. Alle anderen Felder des Kontextes wurden bereits durch das Output-Mapping des *Enter Purchase Request*-Schritts gefüllt und müssen von daher nicht nochmal zugeordnet werden.

Zur Ausführung des Gateways müssen passende Bedingungen für die beiden das Gateway verlassenden Sequenzflüsse gefunden werden. Damit die Ausführbarkeit des Prozesses immer gewährleistet bleibt, verlangt SAP NetWeaver BPM stets einen Pfad, der per Default ausgeführt wird. Diesem Pfad ist immer dann zu folgen, wenn zur Laufzeit kein anderer Pfad aus dem Gateway gefunden werden kann, dessen Bedingungen zutreffen. Im Prozessmodell ist der Default-Pfad (in BPMN auch mit *Standardfluss* bezeichnet) an einem kreuzenden Schrägstrich an dem betroffenen Pfad zu erkennen (siehe auch Abbildung 47). Offensichtlich ist in diesem konkreten Beispiel im Falle einer Ablehnung des Antrags dem Standardfluss zu folgen. Die Bedingung für die Antragsannahme ergibt sich nun wiederum aus dem Zustand des Prozesskontextfeldes *approved*. Nimmt es den booleschen Wert *true* an, so soll entsprechend am Gateway verzweigt werden. Im Process Composer kann dazu in Form eines Scripts eine passende Bedingung hinterlegt werden. Dafür stehen dem Modellierer sämtliche Felder des Kontextes zur Verfügung. Für den Beispielprozess braucht lediglich das

Feld selbst als Bedingung angegeben zu werden, da es sich dabei bereits um eine boolesche Variable handelt (Abbildung 47). Der Editor erlaubt darüber hinaus die Formulierung weitaus komplexerer Ausdrücke, auch unter Verwendung vordefinierter Funktionen.

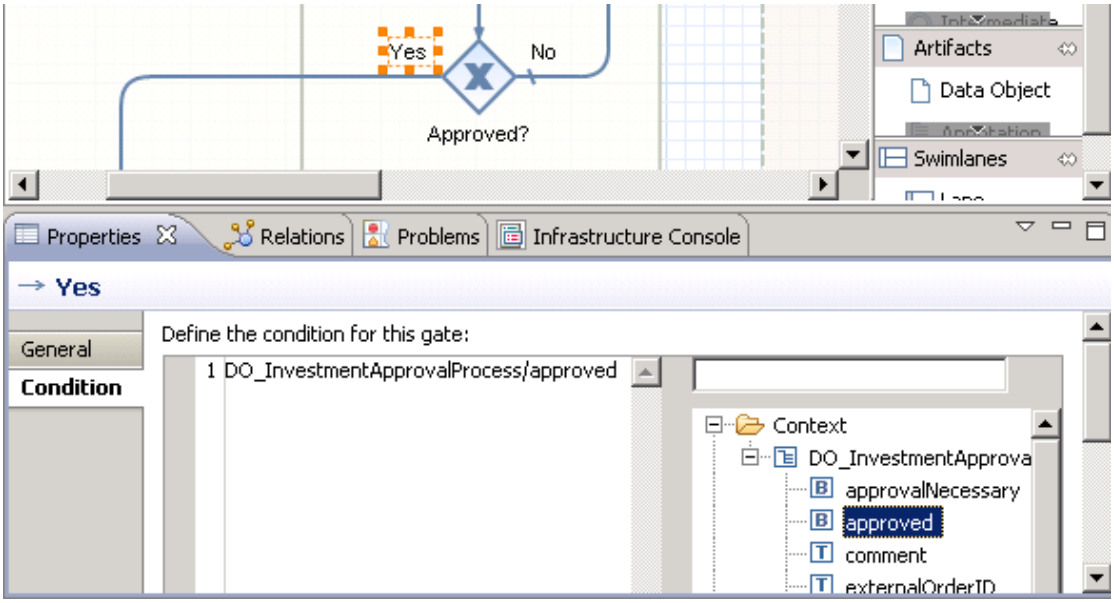



***Abbildung 47: Standardfluss und Bedingung für das exklusive Gateway***

Bleibt zum Schluss die Erläuterung des Zusammenspiels zwischen den beiden Prozessen. Wie bereits in Abschnitt 4.2.3.3 erläutert, koordinieren sich fachlicher und technischer Prozess über gemeinsame asynchrone Schnittstellen. Im Schritt *Create Purchase Order* des fachlichen Prozesses wird die Schnittstelle *createPOOperation* aus Abschnitt 4.2.3.3 verwendet. Dieselbe Schnittstelle ist im technischen Prozess für das Startereignis zu verwenden. Auf diese Weise wird die Verbindung zwischen den beiden Prozessen hergestellt. Analog verläuft die Benachrichtigung des fachlichen Prozesses über die erfolgreiche Verbuchung der Auftragsdaten im Backend-System aus dem technischen Prozess heraus. Der technische Prozesse benutzt dazu in seinem *Confirm PO Creation*-Schritt die *confirmPOCreationOperation*-Schnittstelle. Analog ist im fachlichen Prozess im Nachrichten-Zwischenereignis *Wait for Confirmation* dieselbe Schnittstelle anzugeben. Allerdings reicht diese Zuordnung diesmal allein nicht aus. Schließlich könnten mehrere Prozessinstanzen gerade auf eine Quittierung des technischen Prozesses warten. Daher muss dem Prozessframework mitgeteilt werden, welche konkrete Instanz durch die eingehende Nachricht aktiviert werden soll. Eine einfache Implementierung könnte natürlich diese Aufgabe an die wartenden Pro-

zesse delegieren: grundsätzlich könnten dann beim Eintreffen von Nachrichten einer bestimmten Schnittstelle *alle* wartenden Instanzen aktiviert werden und diese dann selbst aufgrund bestimmter Kriterien prüfen, ob die Nachricht wirklich für sie bestimmt ist. Allerdings würde dies die Leistungsfähigkeit des Gesamtsystems stark beeinträchtigen, da immer wieder parallel Threads gestartet würden, nur um festzustellen, dass die Nachricht in den meisten Fällen nicht für die jeweilige Prozessinstanz bestimmt ist. Von daher wird in SAP NetWeaver BPM auch ein anderer Ansatz verfolgt: zwar wird bei der Modellierung im Prozessmodell die Korrelationsbedingung, wann also eine Instanz zu aktivieren ist, angegeben. Die eigentliche Aktivierung der Prozessinstanz (oder Prozessinstanzen), für die die Bedingung zutrifft, wird allerdings dem Prozessframework zur Laufzeit überlassen. Die Korrelationsbedingung für das konkrete Beispielszenario ergibt sich aus der zuvor erzeugten internen Reservierungsnummer, die ebenfalls dem technischen Prozess übergeben wurde und die ebenfalls Bestandteil der *confirmPOCreationOperation*-Schnittstelle ist. Abbildung 48 zeigt die Korrelationsbedingung für das Nachrichten-Zwischenereignis *Wait For Confirmation*.

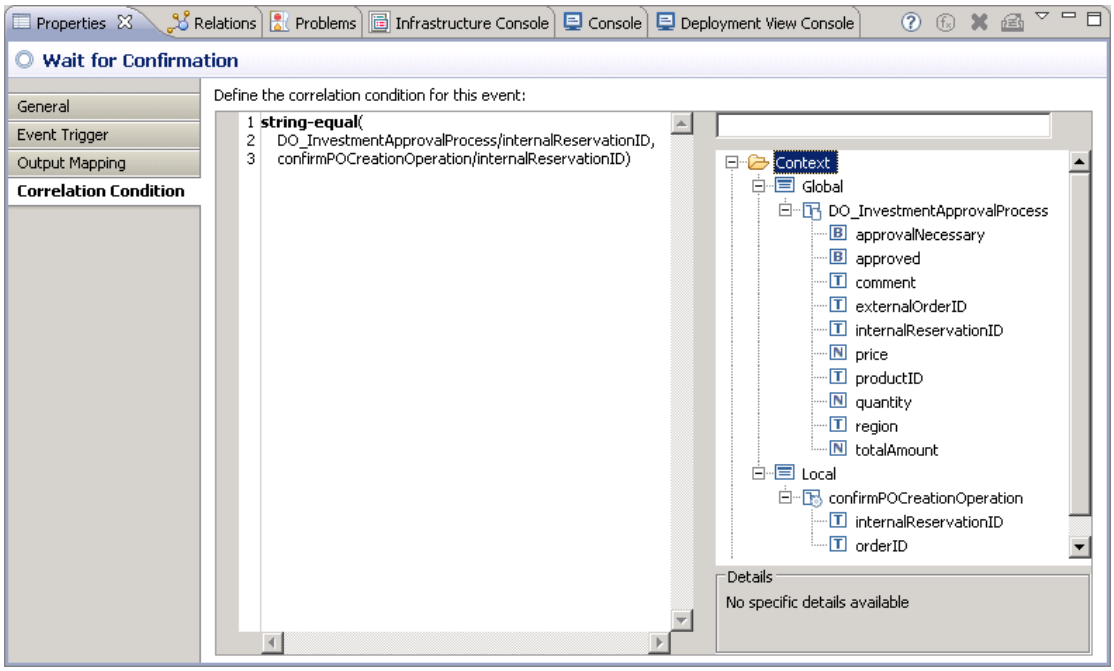



***Abbildung 48: Korrelationsbedingung für das Nachrichten-Zwischenereignis***

Die Bedingung lautet:

```
string-equal(
    DO_InvestmentApprovalProcess/internalReservationID,
    confirmPOCreationOperation/internalReservationID)
```

Wenn also die interne Reservierungsnummer des Prozesskontextes (in der Bedingung referenziert über das Datenobjekt *DO_InvestmentApprovalProcess*) mit der internen Reservierungsnummer, die beim Aufruf der *confirmPOCreationOperation*-Schnittstelle übergeben wird, identisch ist (`string-equal`), dann ist die dazugehörige Prozessinstanz zu aktivieren.

Zusammenfassend lassen sich folgende wesentlichen Aspekte der Implementierung eines lose gekoppelten Szenarios mit SAP NetWeaver BPM festhalten:

- SAP NetWeaver BPM ermöglicht das Zusammenspiel zwischen fachlichen und technischen Prozess über standardisierte asynchrone Web Service-Schnittstellen
- Die Korrelationsbedingungen erlauben dem Prozessframework, die ressourcenschonende Aktivierung von wartenden Prozessinstanzen sicherzustellen.
- Der Datenfluss selbst erfolgt über Input- und Output-Mappings zwischen Aktivitäten und Prozesskontext, der wiederum durch Verwendung des BPMN-Standardelements *Datenobjekt* modelliert wird.
- Die einzelnen Aktivitäten werden mittels Web-Oberflächen für Benutzeraufgaben und mittels Web Services für Serviceaufgaben implementiert. Nachrichten-Startereignisse und Nachrichten-Zwischenereignisse sind ebenfalls mit Web Service-Schnittstellen versehen.
- Die Bestimmung von konkreten Bearbeitern der Aufgaben erfolgt über ein Rollenkonzept, bei dem jeder Prozessrolle, die wiederum einer Lane im Prozessmodell zugewiesen ist, konkrete Gruppen/Rollen einer unternehmensspezifischen Benutzerverwaltung zugeordnet werden. Sämtliche Aktivitäten dieser Lane werden dann durch die derart ermittelten Anwender ausgeführt.

#### 4.2.3.7 Übersichtsdarstellung im Composite Designer

Da Verbundanwendungen, wie gesehen, aus einer Vielzahl unterschiedlichster Komponenten bestehen, ist eine kompakte Darstellung der Beziehungen zwischen den einzelnen Komponenten für den Entwickler von großer Bedeutung. Die Entwicklungsumgebung des SAP NetWeaver Composition Environments sieht dafür eine de-

dizierte Perspektive im SAP NetWeaver Developer Studio vor. Es handelt sich dabei um den Composite Designer. In Abbildung 49 ist die Übersichtsdarstellung für den fachlichen Prozess dargestellt. Darin sind die Verbindungen zu den jeweils direkt vom selektierten Prozess erreichbaren Komponenten gut durch Pfeile zu erkennen.

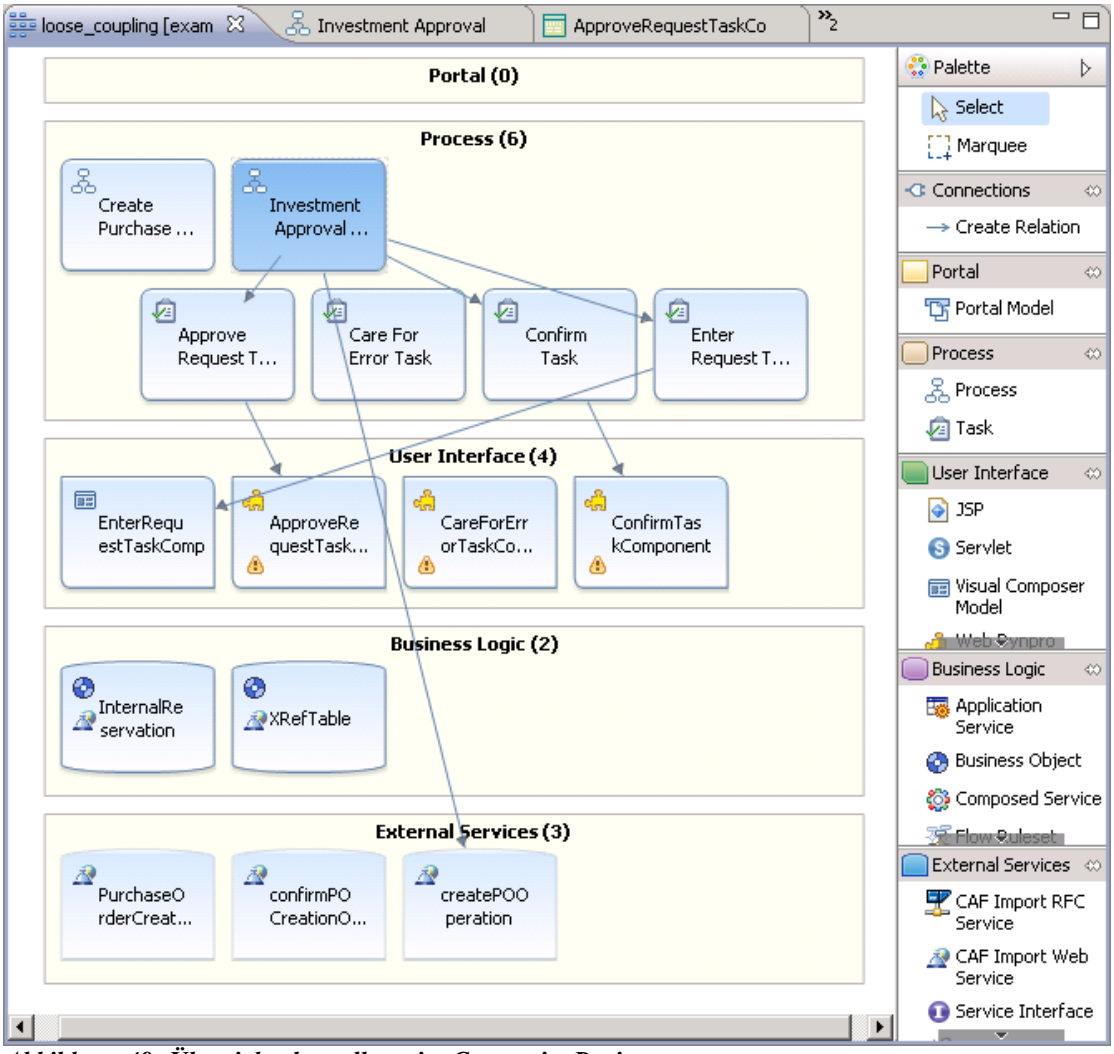



***Abbildung 49: Übersichtsdarstellung im Composite Designer***

Auffällig ist die Aufteilung der Verbundanwendung in die Schichten *Process*, *User Interface*, *Business Logic* und *External Services*. Sie entspricht damit der Architektur von Verbundanwendungen, wie sie auch im Rahmen dieser Arbeit diskutiert wird.

Der Composite Designer dient allerdings nicht nur der Darstellung der Abhängigkeiten zwischen den Komponenten. Er dient auch als Ausgangspunkt zur Anlage neuer Komponenten. So können beispielsweise neue Geschäftsobjekte durch das Ziehen des *Business Object*-Symbols aus der Palette in die *Business Logic*-Schicht angelegt

werden. Der Composite Designer stellt somit das zentrale Cockpit für die Entwicklung von Verbundanwendung innerhalb des Composition Environments dar.

Dieselbe Composite Designer-Darstellung für den technischen Prozess zeigt Abbildung 50.

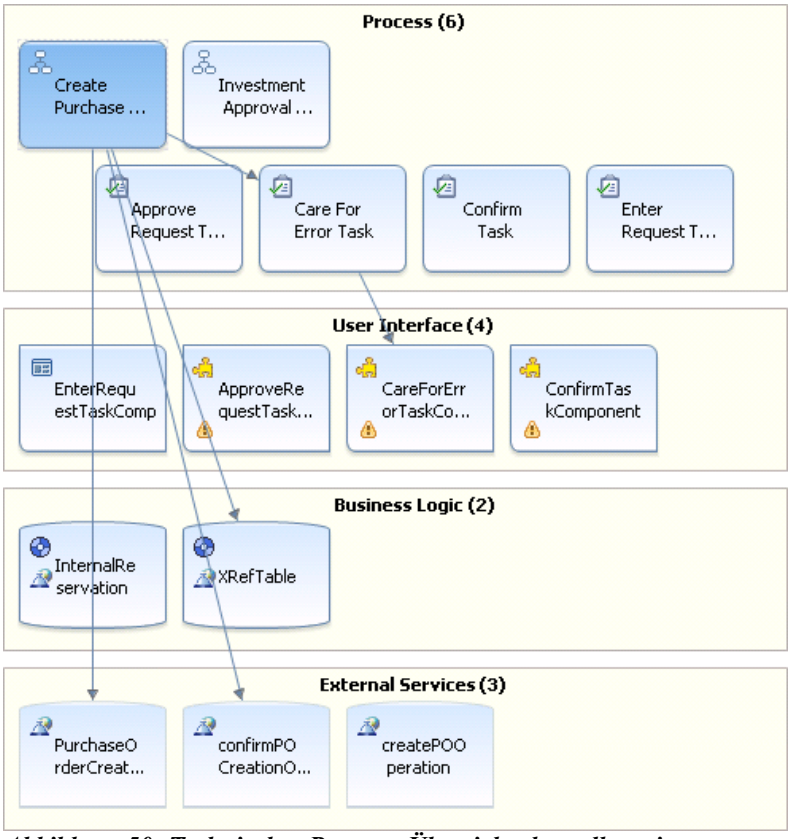



***Abbildung 50: Technischer Prozess: Übersichtsdarstellung im Composite Designer***

Die Abbildung zeigt sehr schön den Aufruf der lokalen Persistenz für die Cross-Referenztabelle als auch die Aufrufe an das Backend-System und an den wartenden fachlichen Prozess über die *confirmPOCreationOperation*-Schnittstelle. Den Aufruf der lokalen Persistenz aus dem Visual Composer-UI heraus zeigt Abbildung 51.

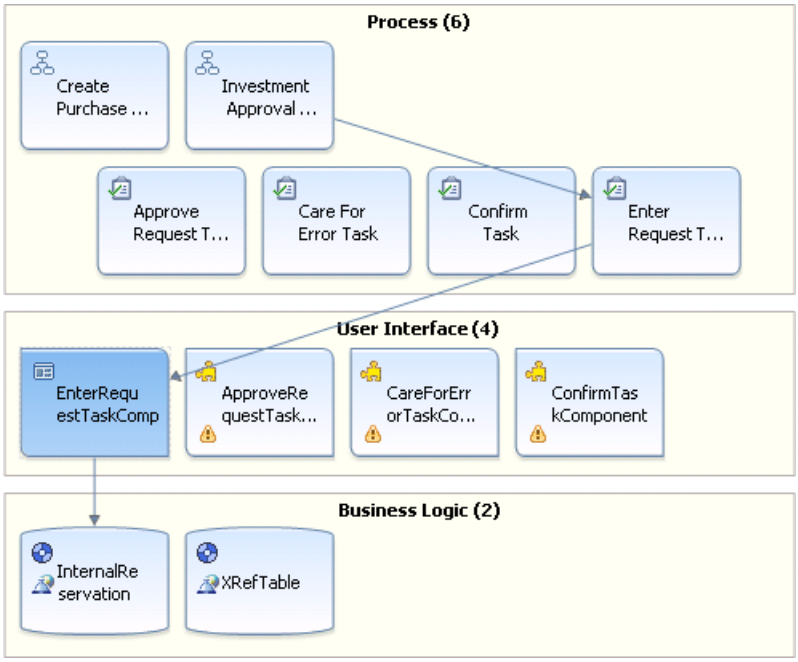



***Abbildung 51: Aufruf der lokalen Persistenz veranschaulicht im Composite Designer***

Sie wird sichtbar, sobald die Visual Composer-Komponente im Composite Designer selektiert ist. Dabei wird gleichzeitig der gesamte Pfad visualisiert, der darstellt, wie das UI in der Verbundanwendung eingebunden ist. Auf diese Weise lassen sich Zusammenhänge innerhalb der Composite Application leicht erkennen und wird es Entwicklern ermöglicht, sich schnell in einer Verbundanwendung zu orientieren.

### *4.2.4 Laufzeitverhalten des Beispielszenarios*

Sämtliche Komponenten der Verbundanwendung lassen sich mit einem Klick auf den JavaEE-basierten Server deployen. Nach dem Deployment und der Ausführung diverser administrative Schritte, bei denen die im Modell verwendeten Service-Schnittstellen mit konkreten Implementierungen verknüpft werden, kann die Anwendung durch Instanziierung des fachlichen Prozesses gestartet werden. Gemäß des modellierten Prozessflusses wird zunächst der Antragsteller über seine Prozessteilnahme in Form einer Benachrichtigung informiert. Diese Benachrichtigung kann in Form einer E-Mail erfolgen. Alternativ wird im Rahmen eines Portals eine zentrale Arbeitsliste zur Verfügung gestellt, in der prozessspezifische Aufgaben erscheinen (siehe auch Abbildung 52). Diese Liste repräsentiert letztendlich den aktuellen Arbeitsvorrat des im Portal angemeldeten Benutzers.

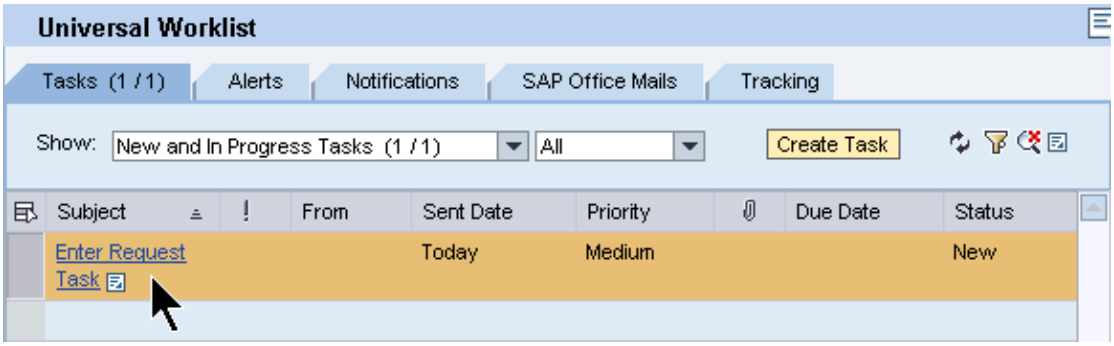



*Abbildung 52: Aufgaben-Vorratsliste*

Der Antragsteller kann durch Klick auf den bereitgestellten Link aus der Liste direkt zur aktiven Aufgabe verzweigen. Es erscheint die der Aktivität zugeordnete Benutzeroberfläche zur Eingabe der Bestelldaten (Abbildung 53).

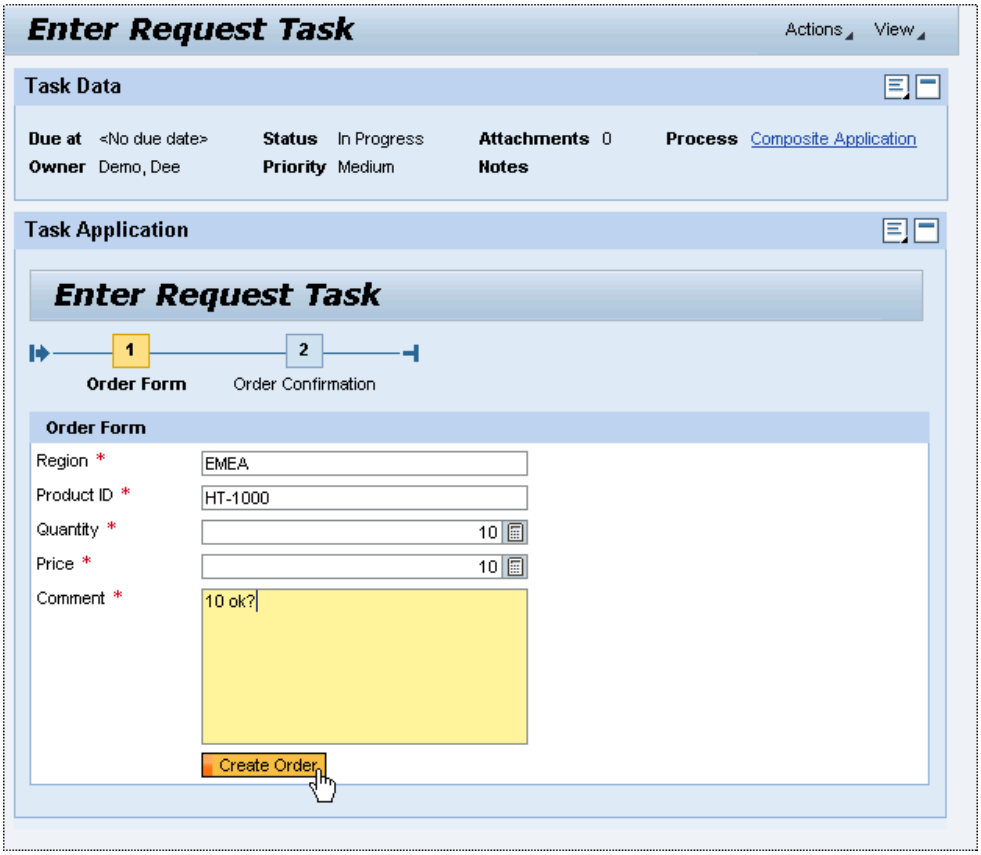



*Abbildung 53: Eingabe der Bestelldaten zur Laufzeit*

Dabei handelt es sich um das mit dem Visual Composer modellierte Wizard-UI, in dessen ersten Schritt die eigentlichen Bestelldaten eingegeben werden. Durch Betätigung des *Create Order*-Buttons wird im Hintergrund der Web Service zur lokalen Per-

sistenz aufgerufen. Diese generiert die interne Reservierungsnummer, die dem Anwender wiederum als Bestätigung der erfolgreichen Dateneingabe im zweiten Schritt des Wizards angezeigt wird (Abbildung 54).

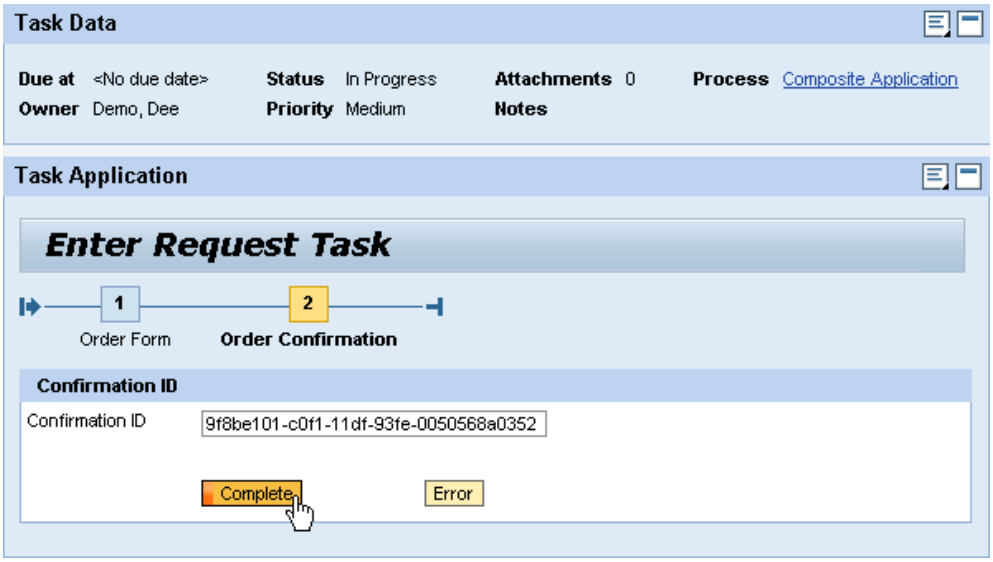



***Abbildung 54: Bestätigung der Reservierung zur Laufzeit***

Der Antragsteller beendet seinen Schritt nun durch Betätigung des *Complete*-Buttons. Dadurch wird ein Ereignis an die Laufzeit des Prozessframeworks geschickt, das nun die eingegeben Daten entgegennehmen und dem Prozesskontext zuweisen kann. Die Aufgabe wird aus dem Arbeitsvorrat des Antragstellers entfernt. Gemäß des Prozessflusses ist als nächstes der Genehmigungsschritt zu aktivieren. Dazu wird eine Benachrichtigung an den der Prozessrolle *Approver* zugewiesenen Personenkreis verschickt, die wiederum in deren Arbeitsvorrat erscheint. Wie bereits erläutert, navigiert der Endanwender per Link direkt zur Aufgabe. Die dazugehörige Oberfläche stellt die vom Antragsteller eingegebenen Daten dar, die zuvor vom Prozessframework aus dem Prozesskontext gemäß Input-Mapping-Vorschrift versorgt wurden (Abbildung 55).

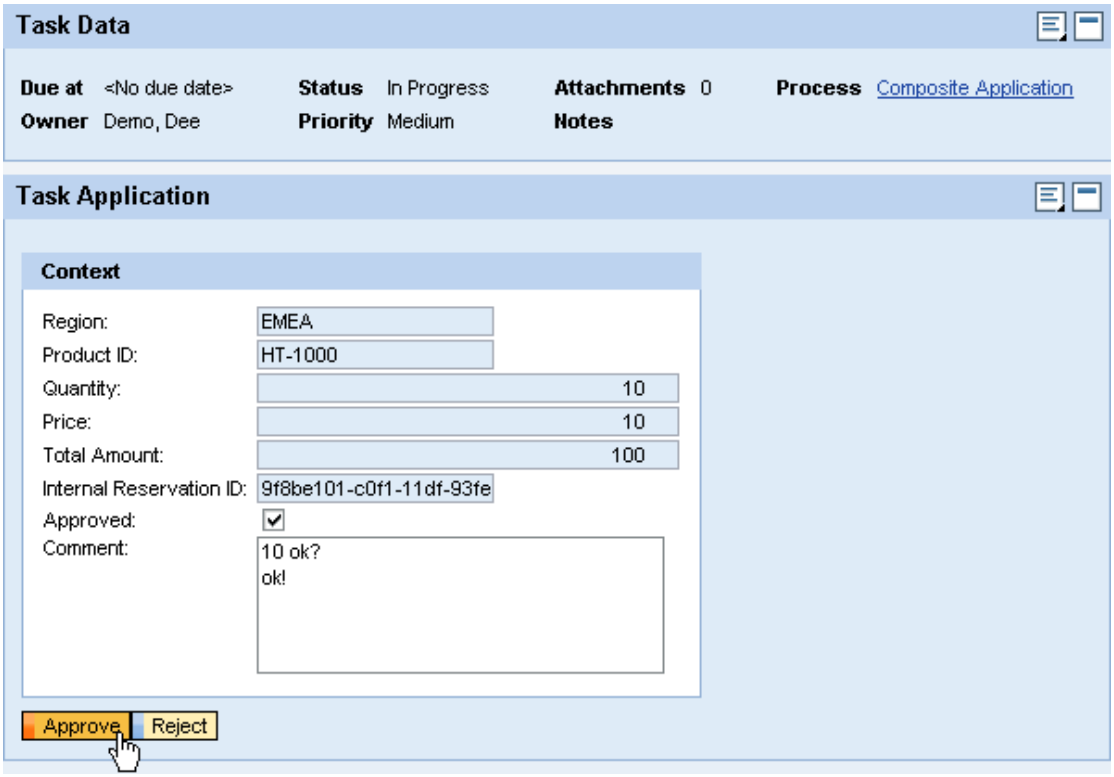



*Abbildung 55: Genehmigungsschritt zur Laufzeit*

Der Genehmigende bestätigt die Bestellung durch Setzen der *Approved*-Checkbox und schließt den Schritt durch Betätigung des *Approve*-Buttons ab. Die veränderten Daten gelangen daraufhin in den Prozesskontext.

Nun erfolgt im Hintergrund das Zusammenspiel zwischen Verbundanwendung und Servicevertrag-Implementierungsschicht. Die Verbundanwendung startet den technischen Prozess durch Übergabe von Produktkennung, Bestellmenge sowie interner Reservierungsnummer und wartet anschließend am nachrichtenbasierten Zwischenereignis. Der technische Prozess ruft das Backend-System zur Verbuchung der Bestelldaten auf, um anschließend die interne Reservierungsnummer mit der im Backend erzeugten externen Nummer in der Cross-Referenztabelle abzulegen. Anschließend ruft sie den Web Service auf, der der Schnittstelle des Zwischenereignisses entspricht, an dem der Prozess der Verbundanwendung wartet. Dieser nimmt nach der Reaktivierung nun seinerseits die extern erzeugte Bestellnummer in den Prozesskontext auf, von wo sie dem Antragsteller in seinem Bestätigungsschritt angezeigt werden kann (Abbildung 56).

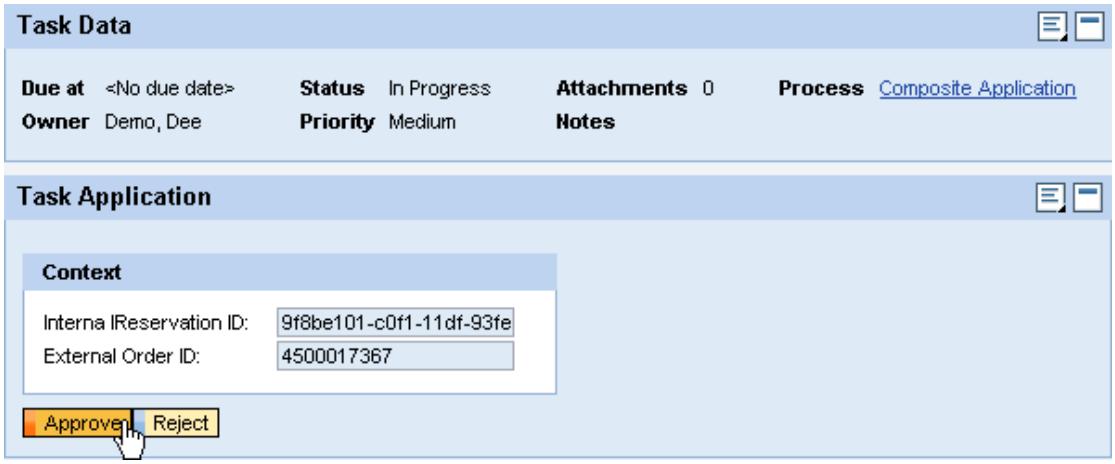



*Abbildung 56: Bestätigung der asynchronen Verbuchung zur Laufzeit*

Da die Service-Aufgaben des technischen Prozesses im Hintergrund ohne menschliche Interaktionen abliefen, können lediglich die Log-Einträge sowie der neue Eintrag in der Cross-Referenztabelle das korrekte Zusammenspiel der beiden Prozesse dokumentieren. Abbildung 57 zeigt daher den neu erzeugten Eintrag in der Cross-Referenztabelle und die Abbildungen 58 sowie 59 die Details der Prozessausführung von sowohl betriebswirtschaftlichen als auch technischen Prozess.

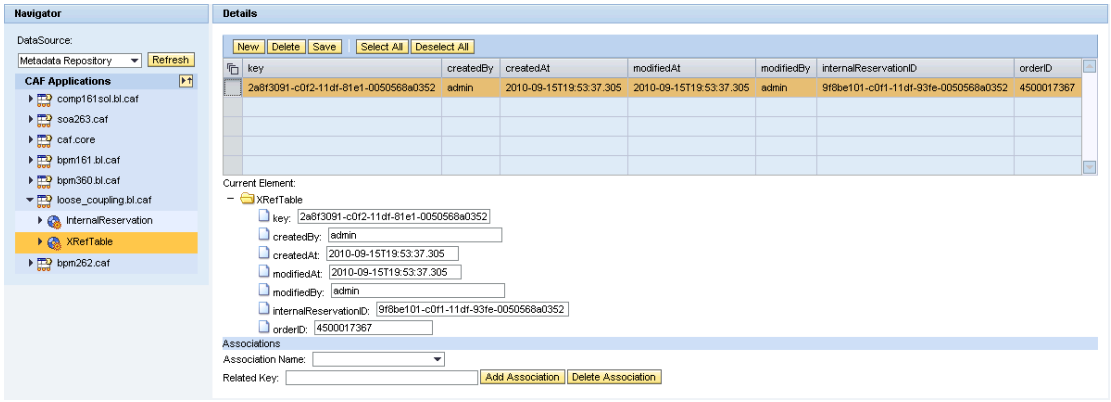
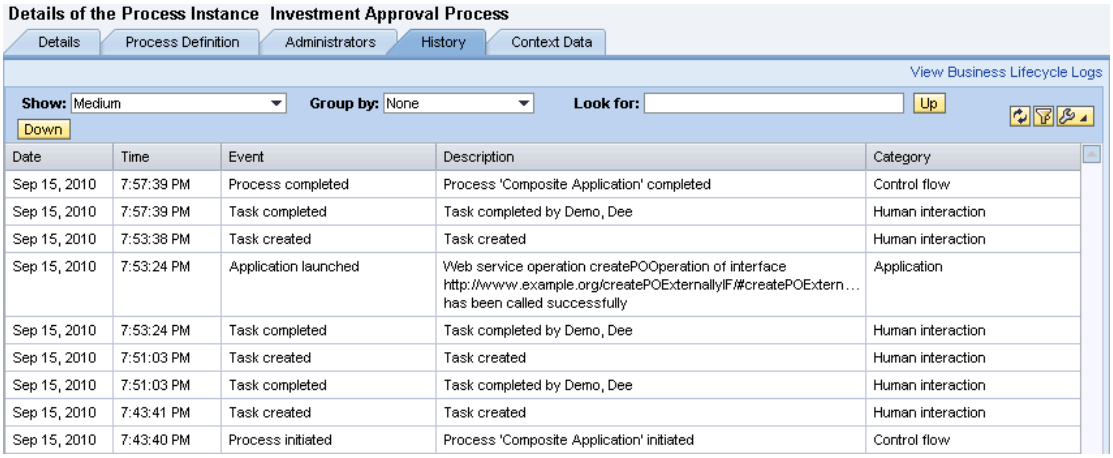
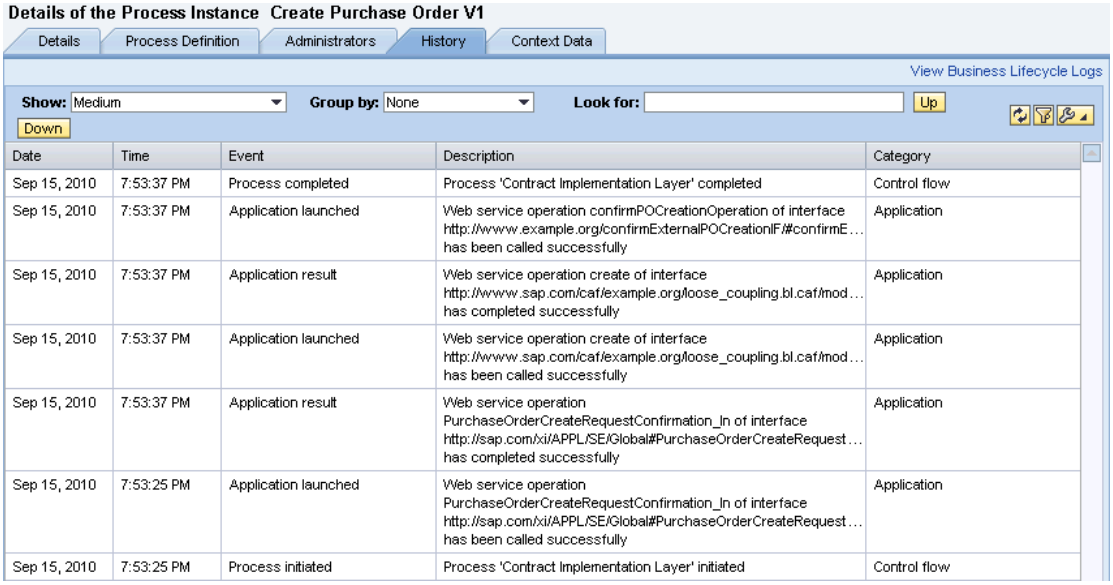



*Abbildung 57: Eintrag in Cross-Referenztabelle nach Beendigung des Prozesses*




| Date | Time | Event | Description | Category |
|---|---|---|---|---|
| Sep 15, 2010 | 7:57:39 PM | Process completed | Process 'Composite Application' completed | Control flow |
| Sep 15, 2010 | 7:57:39 PM | Task completed | Task completed by Demo, Dee | Human interaction |
| Sep 15, 2010 | 7:53:38 PM | Task created | Task created | Human interaction |
| Sep 15, 2010 | 7:53:24 PM | Application launched | Web service operation createPOOperation of interface http://www.example.org/createPOExternallyIF/#createPOExtern… has been called successfully | Application |
| Sep 15, 2010 | 7:53:24 PM | Task completed | Task completed by Demo, Dee | Human interaction |
| Sep 15, 2010 | 7:51:03 PM | Task created | Task created | Human interaction |
| Sep 15, 2010 | 7:51:03 PM | Task completed | Task completed by Demo, Dee | Human interaction |
| Sep 15, 2010 | 7:43:41 PM | Task created | Task created | Human interaction |
| Sep 15, 2010 | 7:43:40 PM | Process initiated | Process 'Composite Application' initiated | Control flow |

***Abbildung 58: Prozessausführung betriebswirtschaftlicher Prozess***




| Date | Time | Event | Description | Category |
|---|---|---|---|---|
| Sep 15, 2010 | 7:53:37 PM | Process completed | Process 'Contract Implementation Layer' completed | Control flow |
| Sep 15, 2010 | 7:53:37 PM | Application launched | Web service operation confirmPOCreationOperation of interface http://www.example.org/confirmExternalPOCreationIF/#confirmE… has been called successfully | Application |
| Sep 15, 2010 | 7:53:37 PM | Application result | Web service operation create of interface http://www.sap.com/caf/example.org/loose_coupling.bl.caf/mod… has completed successfully | Application |
| Sep 15, 2010 | 7:53:37 PM | Application launched | Web service operation create of interface http://www.sap.com/caf/example.org/loose_coupling.bl.caf/mod… has been called successfully | Application |
| Sep 15, 2010 | 7:53:37 PM | Application result | Web service operation PurchaseOrderCreateRequestConfirmation_In of interface http://sap.com/xi/APPL/SE/Global#PurchaseOrderCreateRequest… has completed successfully | Application |
| Sep 15, 2010 | 7:53:25 PM | Application launched | Web service operation PurchaseOrderCreateRequestConfirmation_In of interface http://sap.com/xi/APPL/SE/Global#PurchaseOrderCreateRequest… has been called successfully | Application |
| Sep 15, 2010 | 7:53:25 PM | Process initiated | Process 'Contract Implementation Layer' initiated | Control flow |

***Abbildung 59: Prozessausführung technischer Prozess***

Die Logging-Einträge sind dabei von unten nach oben zu lesen. Im Logging des betriebswirtschaftlichen Prozesses sind zunächst die beiden Benutzeraufgaben zu erkennen, ehe um 7:53:24 Uhr durch einen asynchronen Aufruf die Instanziierung des technischen Prozesse erfolgt. Asynchrone Aufrufe sind an nur einem Log-Eintrag zu erkennen, während sich synchrone Aufrufe durch zwei Log-Einträge auszeichnen. Der erste Logging-Eintrag des technischen Prozesses bestätigt seine Instanziierung durch den Zeitstempel 7:53:24 Uhr, der sich zeitlich gesehen nahtlos dem Aufruf des betriebswirtschaftlichen Prozesses anschließt. Die erste Aktion des technischen Prozesses besteht im synchronen Aufruf des SAP-Backend-Systems zum Anlegen des Auftrags. Dieser dauert von 7:53:25 bis 7:53:37. Die nächsten beiden Einträge reflektie-

ren das synchrone Schreiben (daher zwei Einträge) in die Cross-Referenztabelle, ehe der wartende Prozess vom technischen Prozess asynchron aus eben dieser Wartesituation befreit wird (7:53:37). Da das Schreiben in die lokale Datenbank erfolgt, vergeht auch nahezu keine Zeit. Der technische Prozess beendet sich unmittelbar nach Reaktivierung des wartenden Prozesses. Da dies asynchron erfolgt, ist auch hier kein Zeitunterschied zur vorangegangenen Aktion erkennbar.

Der betriebswirtschaftliche Prozess setzt wiederum ohne Verzögerung (7:53:38) seine Bearbeitung mit der Erstellung der Benutzeraufgabe für den Antragsteller fort. Auch dieser Prozess beendet sich letztendlich nach Schließen der Bestätigungsmaske durch den Antragsteller.

Die Ausführung des Prozesses ergänzt um das Logging-Protokoll lassen die Vorteile der Architektur gut erkennen: langwierige Backend-Aufrufe sind nun sauber von der Verbundanwendung entkoppelt. Lokale Datenbankaufrufe erfolgen schnell und erlauben dem Endanwender ein flüssiges Arbeiten. Außerdem könnte die Implementierung des externen Zugriffs leicht durch eine Alternativlösung ersetzt und damit die Verbundanwendung flexibel an andere Systemlandschaften angepasst werden. Schließlich bestehen keinerlei Abhängigkeiten zu Datentypen oder proprietären Web Service-Schnittstellen. Diese Vorteile wurden durch eine leicht erhöhte Komplexität erkauft, die allerdings durch moderne Entwicklungsumgebungen zumindest gelindert werden kann.

### *4.2.5 Rolle der modellgetriebenen Entwicklung*

Die Beispielimplementierung unterstreicht die Umsetzbarkeit des vorgeschlagenen Architekturansatzes mit Hilfe von lose gekoppelten BPMN-Prozessen. Durch die Verwendung einer modernen modellgetriebenen Entwicklungsumgebung konnte der Aufwand auf ein Minimum reduziert werden. Konkret musste zur Implementierung dieses Proof-of-Concepts nicht eine Zeile Programmcode entwickelt werden. Besonders aufwändige und fehleranfällige Arbeiten wie die Ausgestaltung von Benutzeroberflächen, die Persistenz von Daten, die Programmierung von Nebenläufigkeit oder auch die Kopplung verschiedener Systeme über Service-Schnittstellen können durch modellgetriebene Ansätze vereinfacht werden. Zudem sind dadurch nun genau die Architekturen effizient umsetzbar, die zuvor aufgrund des Aufwands gemieden wurden. Dabei sorgen die BPMN-Elemente des *nachrichtenbasierten Zwischenereignisses* bzw. alternativ der *empfangenden Aufgabe* für eine relativ unkomplizierte Umsetzung lose ge-

koppelter Architekturen. Sämtliche Hersteller, die die BPMN-Spezifikation realisieren, müssen in ihren Frameworks entsprechende Lösungen für diese Elemente umsetzen und damit eine Infrastruktur für die Prozesskooperation bereitstellen. Damit stellt die Modellierung und Ausführung von Prozess-Föderationen eine besondere Stärke von BPMN dar.

## 4.3 Bedeutung der BPMN-Implementierung verschiedener Hersteller

Das nachrichtenbasierte Zwischenereignis übernimmt in der Umsetzung des lose gekoppelten Szenarios eine zentrale Funktion. Nach Start des technischen Prozesses wird es im betriebswirtschaftlichen Prozess dazu benutzt, die Wartefunktion umzusetzen. Daraus wird er erst wieder befreit, sobald die passende Nachricht vom technischen Prozess eintrifft. Zwar sieht das Prozessmodell dazu so aus, als könnten keine Probleme auftreten, doch bei genauerer Betrachtung ist es denkbar, dass eine Deadlock-Situation entsteht. Ob eine Deadlock-Situation entsteht oder nicht hängt davon ab, wie das nachrichtenbasierte Zwischenereignis vom Hersteller der BPMN-Implementierung umgesetzt wird.

Es könnte theoretisch passieren, dass die Abarbeitung des Backend-Prozesses schneller erfolgt als das Zwischenereignis aktiv wird, also das Token, das durch den Prozess wandert, das Zwischenereignis erreicht. Dann stellt sich zwangsläufig die Frage, was mit der Nachricht passieren soll, die vom technischen Prozess gesendet wird, aber auf ein noch nicht in Bereitschaft befindliches Zwischenereignis trifft. Noch offensichtlicher wird das Problem, wenn der Nachrichtenempfang zeitlich überwacht werden soll und so wie in Abbildung 60 modelliert wurde.

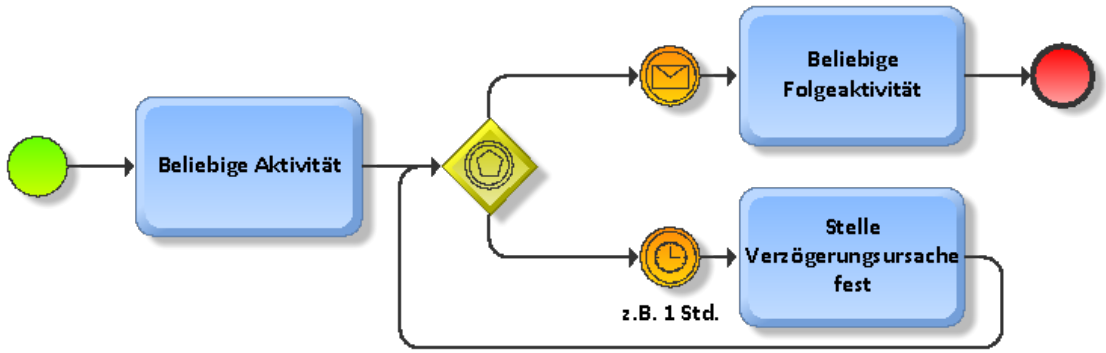



*Abbildung 60: Zeitüberwachung des Nachrichtenempfangs über ein ereignisbasiertes Gateway*

Tritt nun das zeitbasierte Zwischenereignis vor dem Eintreffen der Nachricht ein, so kann sich das Token entweder auf dem zeitbasierten Zwischenereignis oder schon auf der danach folgenden Aktivität *Stelle Verzögerungsursache fest* befinden. Was passiert aber dann mit der Nachricht, die genau zu einem solchen Zeitpunkt eintrifft und sich folglich kein Token auf dem nachrichtenbasierten Zwischenereignis befindet? In der BPMN 2.0-Spezifikation (OMG 2010a) heißt es dazu im Kapitel über die Ausführungssemantik:

S. 439: „An Activity is Ready for execution if the REQUIRED number of tokens is available to **activate** the Activity. The REQUIRED number of tokens (one or more) is indicated by the attribute StartQuantity."

S. 440: "Receive Task: Upon **activation**, the Receive Task **begins waiting** for the associated Message. When the Message arrives, the data in the Data Output of the Receive Task is assigned from the data in the Message, and Receive Task completes."

S. 451: "For Intermediate Events, the handling consists of waiting for the Event to occur. **Waiting starts when the Intermediate Event is reached**."

Aus der Spezifikation geht demnach lediglich hervor, ab wann Nachrichten empfangen werden können, nämlich dann, wenn das Token entweder die Empfangs-Aufgabe oder das empfangende nachrichtenbasierte Zwischenereignis erreicht. Sie sagt aber nichts darüber aus, was mit Nachrichten geschehen soll, die eintreffen, obwohl weder eine Receive-Task noch ein Zwischenereignis bereit ist. Auch Silver 2009 beschreibt das Verhalten am ereignisbasierten Gateway wie folgt:

S. 90: „The event that occurs first determines the path enabled out of the gateway. Any trigger signals arriving after that are ignored."

Streng genommen könnte daraus geschlossen werden, dass Nachrichten grundsätzlich verworfen werden, die zu einem Zeitpunkt eintreffen, an dem kein Empfänger bereit ist. Genau dann kann in den oben genannten Fällen zur Laufzeit eine Deadlock-Situation eintreten. SAP verfolgt in ihrer Implementierung des nachrichtenbasierten Zwischenereignis diesen Ansatz nicht (siehe Balko 2010a). Sämtliche Nachrichten, die die Korrelationsbedingung erfüllen, werden an dem nachrichtenbasierten Zwi-

schenereignis in einer Warteschlange eingereiht, auch wenn noch kein Token am Zwischenereignis wartet. Nachrichten werden der Warteschlange zu dem Zeitpunkt entnommen, sobald ein Token das dazugehörige Zwischenereignis erreicht. Auf diese Weise ist eine deadlockfreie Verarbeitung von Nachrichten möglich und deren Verlust ausgeschlossen.

Für Hersteller, die diesem Ansatz nicht folgen und daher Gefahr laufen, Nachrichten zu verlieren, kann zumindest von der BPMN-Modellierung her das in Abbildung 61 dargestellte Modell weiterhelfen.

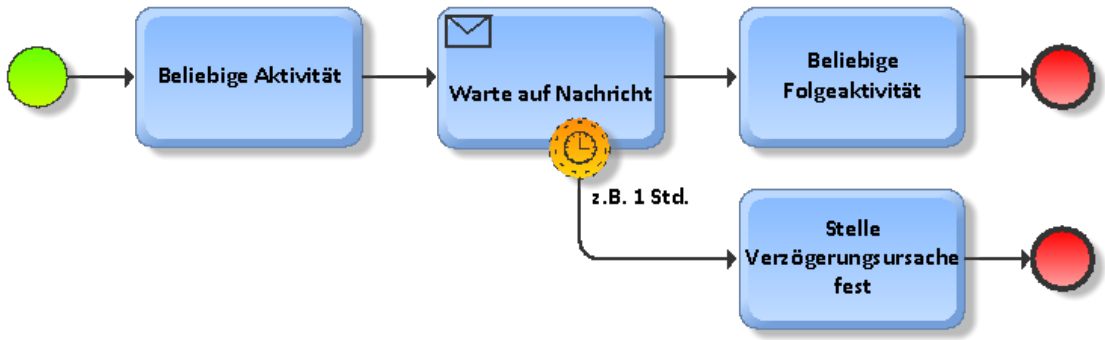



***Abbildung 61: Zeitüberwachung des Nachrichtenempfangs über ein angeheftetes nicht-unterbrechendes Zwischenereignis***

In diesem Prozessmodell wird zumindest das Warten auf Nachrichten nicht durch das angeheftete zeitbasierte Zwischenereignis unterbrochen. Dadurch wird der Nachrichtenverlust während der Ursachenanalyse verhindert. Allerdings löst auch dieses Modell das generelle Problem nicht, das Nachrichten verloren gehen, die zu dem Zeitpunkt eintreffen, wenn sich das Token beispielsweise noch auf der Aktivität vor der Empfangsaufgabe befindet. In der SAP-Implementierung kann auch dann keine Nachricht verloren gehen, sofern die Korrelationsbedingung für die Empfangsaufgabe erfüllt ist. Wie der Beispielimplementierung entnommen werden konnte, wird bei SAP NetWeaver BPM die Korrelationsbedingung durch Felder des Prozesskontextes gebildet. Wird dieser Kontext bereits durch das Prozess-Startereignis gefüllt, so ist dadurch automatisch das nachrichtenbasierte Zwischenereignis aktiviert und Nachrichten können bereits zu diesem frühen Zeitpunkt der Warteschlange hinzugefügt werden.

Anhand dieses einfachen Beispiels konnte verdeutlicht werden, dass ein anscheinend eindeutiges BPMN-Modell zur Laufzeit durchaus zu ungeahnten Problemen füh-

ren kann. Die Kenntnis darüber, wie Hersteller die verschiedenen BPMN-Elemente implementiert haben, ist daher unerlässlich.

## 4.4 Versionsmanagement

Die in den Verbundanwendungen abgewickelten Geschäftsprozesse erstrecken sich im Allgemeinen über einen längeren Zeitraum. In Extremfällen kann ein Prozess über mehrere Jahre aktiv bleiben. Auch wenn nicht immer von solchen Extremen ausgegangen werden kann, so sind Laufzeiten von Wochen und Monaten schon keine Seltenheit mehr. Damit verbunden sind zwangsläufig Fragen nach dem Verhalten dieser Prozesse, wenn neue Versionen bereitgestellt werden: neue Versionen der Prozesse selbst sowie neue Versionen der in den Prozessen referenzierten Inhalte, also Service-Aufrufe und Benutzeroberflächen.

Das Problem kann für referenzierte Inhalte über eine geeignete Namensgebung gelöst werden: die Versionsnummer wird Bestandteil des Komponentennamens. Verweist ein Prozess beispielsweise auf eine Benutzeroberfläche zur Auftragseingabe, so wird als Referenz für den dazugehörigen Prozessschritt der Name *Auftragseingabe_V1.1* vergeben. Prozessschritt und UI-Komponente sind über diesen Namen eindeutig verbunden. Soll nun eine aktualisierte Version des UIs zur Verfügung gestellt werden, so muss entschieden werden, ob die neue Version auch für die noch laufenden Prozessinstanzen Gültigkeit besitzen soll. Ist dem so, und es handelt sich lediglich um kosmetische UI-Änderungen wie die Anpassung von Farben an ein neues Corporate Identity oder die Positionierung der Felder innerhalb des UIs, so wird die überarbeitete Version ebenfalls unter dem Namen *Auftragseingabe_V1.1* im Versionsmanagement-System geführt. Lediglich innerhalb der Versionsverwaltung werden die Versionen unterschieden, nicht jedoch zur Laufzeit. Nach dem Deployment unter demselben Namen, werden auch alle laufenden Prozessinstanzen die neue Oberfläche aufrufen.

Handelt es sich allerdings um eine nicht-kompatible Änderung wie das Hinzufügen von Feldern oder gar neuer Laschen, so ist auch der Name der betroffenen Benutzeroberfläche anzupassen (z.B. *Auftragseingabe_V1.2*), und die Referenzen in den jeweiligen Prozessen, die dann ebenfalls eine neue Versionsnummer erhalten, sind zu aktualisieren. Wird nun eine neue Instanz eines derart aktualisierten Prozesses gestartet, so

läuft dieser mit dem neuen UI, während alle bereits laufenden Prozesse noch mit der alten Oberfläche arbeiten.

Dasselbe Verfahren kann für Service-Aufrufe verwendet werden. Auch hier wird über die Namensgebung gesteuert, ob neue Versionen eines Service auch für bereits aktivierte Prozessinstanzen verwendet werden sollen oder nicht.

Bei Prozessveränderungen gilt in der Regel das Verhalten, dass bereits laufende Prozesse auch mit dem Prozessmodell beendet werden, mit dem sie gestartet wurden. Sollen jedoch Veränderungen nachträglich auch an bereits laufenden Prozessinstanzen vorgenommen werden, so müssen die solche Szenarien unterstützenden BPM-Systeme von ihrer Architektur her darauf vorbereitet sein: sie erlauben Ad-Hoc-Veränderungen bei gleichzeitiger Gewährleistung der Ausführbarkeit und der Robustheit aller folgender Schritte selbst nach der Anpassung durch entsprechende Überprüfungen. Auch sogenannte Prozess-Schema-Evolutionen, also Migrationen laufender Prozessinstanzen hin zu neuen Prozessmodellen, sind dabei zu berücksichtigen. Einen guten Überblick über Forschungsarbeiten in diesem Bereich liefert Dadam et. al. 2009 bzw. das ADEPT2-Forschungsprojekt der Universität Ulm (ADEPT2 2010).

## 4.5 Komponenten als Voraussetzung für Verbundanwendungen – die Rolle von Varianten-Komponenten-Diagrammen

Die bisherigen Ausführungen haben bereits deutlich die Komponentisierung von Verbundanwendungen herausgearbeitet. Betrachtet man die verschiedenen Schichten der Composite, so gibt es keine Ebene, bei der sie nicht zum Einsatz kommt. Auf Prozessebene können aufgrund der Trennung Verbundanwendung – Servicevertrag-Implementierungsschicht und der fest vorgegebenen, standardisierten Schnittstelle zur Umsetzung einer fachlichen Anforderung der Composite Application je nach Anwendungsfall und Kontext verschiedene Implementierungen der Fachlichkeit flexibel zum Einsatz kommen. Es gibt also verschiedene Varianten für die Implementierung. Dasselbe gilt für die Benutzeroberflächen-, Service- oder Geschäftsregelkomponenten. Sie alle sind aufgrund der Anforderung der losen Kopplung flexibel austauschbar gestaltet. Da es sich bei Verbundanwendungen um webbasierte Lösungen handelt, sind Benutzeroberflächen über eindeutige URLs (Uniform Resource Locator) zu erreichen. Hinter einer solchen URL kann natürlich jedes beliebige UI eingesetzt werden, solange sie die spezifizierte Funktionalität erbringt. Dasselbe gilt für Services oder Geschäftsregeln, die als Web Services bereitgestellt werden. Sie sind ebenfalls über ihre

eindeutigen Endpunkte adressierbar. Die Implementierung selbst lässt sich aber auch hier durch die Komponentisierung austauschen. Unterschiedliche Varianten können somit in unterschiedlichen Kontexten zum Einsatz kommen. Komponenten spielen für Verbundanwendungen folglich eine fundamentale Rolle, die den Flexibilitäts- und Anpassbarkeitsaspekt unterstützen. In diesem Sinne ist eine Verbundanwendung also auch eine komponentenbasierte Anwendung (Ortner 2005a). Damit verbunden ist allerdings die Fragestellung, wie die Gesamtanwendung konfiguriert werden kann, wenn mehrere Varianten für eine spezifische Funktionalität zur Verfügung steht?

Prinzipiell lassen sich zwei Konfigurationsarten unterscheiden: die dynamisch maschinelle und die statisch manuelle. Bei der statisch manuellen Konfiguration werden die Komponenten in einem interaktiven Konfigurationsschritt durch eine dedizierte Rolle, dem Konfigurator, zusammengefügt. Im Gegensatz dazu ermittelt bei der dynamisch maschinellen Vorgehensweise der Rechner zur Laufzeit die passende Komponente. Umgesetzt werden kann die letztgenannte Methode durch den Einsatz von Regelwerken, wie dies in Kapitel 5.7 beschrieben wird.

Für die statisch manuelle Variante beschreibt Grollius 2010 eine Vorgehensweise zur interaktiven Konfigurierung dynamischer Anwendungssysteme, die sich aus Komponenten zusammensetzen. Der Kerngedanke des Papiers ist die Visualisierung der Aufbaustruktur eines komponentenbasierten Anwendungssystems in Form eines sogenannten Varianten-Komponenten-Diagramms (VCD – Variant Component Diagram) sowie die werkzeugunterstützte interaktive Konfigurierung des Gesamtsystems auf Basis des VCDs. Bevor es also zur eigentlichen Konfiguration des Gesamtsystems kommen kann, muss dessen Aufbau zuvor geeignet beschrieben werden können. Dafür eignet sich das VCD, das ursprünglich auf den Mereologischen Graphen als grafische Repräsentation von Variantenstrukturstücklisten basiert. Die mereologischen Graphen wurden von Wedekind/Müller bereits 1981 vorgestellt und von Hümmer 2004 erweitert (Wedekind und Müller 1981; Hümmer 2004). Die aus dem Papier von Grollius stammende Abbildung 62 veranschaulicht an einem einfachen Beispiel die grundlegende Idee eines Varianten-Komponenten-Diagramms und der ihr zugrundeliegenden Varianten-Komponenten-Diagrammsprache.

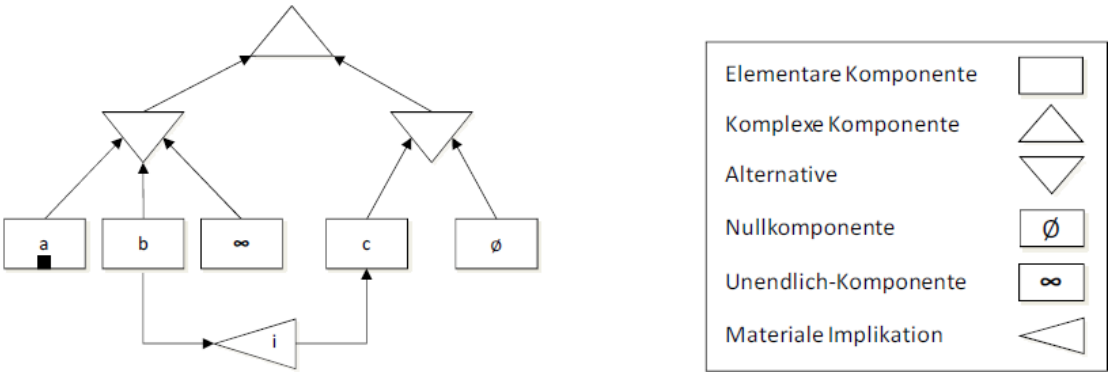



***Abbildung 62: Varianten-Komponenten-Diagramm (aus Grollius 2010)***

Im Wesentlichen werden durch das Diagramm Teil-Ganzes-Beziehungen dargestellt, wobei das Diagramm von der Wurzel an der Spitze nach unten zu lesen ist. Konkret bedeutet das für das Beispiel, dass sich die komplexe Komponente an der Spitze aus zwei Komponenten zusammensetzt. Eine komplexe Komponente repräsentiert dabei eine logische Und-Verknüpfung (Konjunktion). Offensichtlich werden dabei die Teil-Ganzes-Beziehungen in Form von gerichteten Kanten dargestellt, die vom Unterteil zum Oberteil verlaufen und dadurch zum Ausdruck bringen, dass sich das Oberteil aus den Unterteilen zusammensetzt. Im Beispiel setzt sich das Gesamtsystem aus zwei Alternativen zusammen, wobei allerdings nicht gemeint ist, dass entweder die linke *oder* die rechte Alternative genommen werden kann. Die Alternativen laufen in die komplexe Komponente ein und sind damit Und-Verknüpft. Mit den Alternativen wird ausgedrückt, dass die in den Alternativ-Knoten einlaufenden Kanten mit einem logischen exklusiven Oder verknüpft sind. Hierbei handelt es sich folglich genau um die Varianten, die später konfiguriert werden können. Für die linke Alternative können demnach die elementaren Komponenten a, b oder die Unendlich-Komponente ausgewählt werden und für die rechte Alternative die elementare Komponente c oder die Nullkomponente. Elementare Komponenten repräsentieren Knoten, die nicht weiter in Komponenten zerlegt werden können und repräsentieren demnach eine Software-Komponente, die die geforderte Funktionalität erfüllt.

Im Diagramm sind bei den Komponenten weitere Unterscheidungen zu erkennen. Die Elementarkomponente a enthält als spezielle Markierung ein schwarzes Quadrat. Auf diese Weise wird die Standardvariante (Default) gekennzeichnet, die immer dann verwendet wird, wenn der Konfigurator aus den möglichen Alternativen keine Aus-

wahl vornimmt. Die Unendlich-Komponente übernimmt eine spezielle Funktion: an dieser Stelle kann der Konfigurator frei entscheiden, welche Komponente er einsetzen möchte. Bei den Komponenten a und b ist bekannt, dass sie der geforderten Spezifikation folgen. Bei der frei wählbaren Unendlich-Komponente ist dies nicht notwendigerweise erfüllt. Hier kann ein entsprechendes Werkzeug dem Konfigurator helfen, indem er bei der Auswahl nur Komponenten aus einem Komponentenkatalog zulässt, die den Anforderungen genügen. Die Nullkomponente schließlich, die unterhalb der rechten Alternative zu erkennen ist, symbolisiert eine Kann-Alternative. Wird sie ausgewählt, so bringt der Konfigurator dadurch seinen Verzicht auf eine Komponentenauswahl zum Ausdruck.

Im Diagramm fällt zudem die Verbindung der Elementarknoten b und c über eine sogenannte materielle Implikation auf (Wedekind 1989). Mit ihr werden letztendlich Abhängigkeiten zwischen den Varianten dargestellt. Im konkreten Fall muss bei Auswahl der Komponente b auch zwingend c ausgewählt werden. Der Konfigurator hätte also bei Wahl der Komponente b für die rechte Alternative keine Auswahlmöglichkeit mehr, da diese durch die materielle Implikation vorgegeben ist. Umgekehrt gilt, dass b als Auswahl nicht möglich ist, wenn Elementarkomponente c nicht infrage kommt.

Auf Basis der Varianten-Komponenten-Diagrammsprache können nun Komponenten mit ihren Abhängigkeiten, mögliche Varianten für eine Komponente, Default-Varianten, Kann-Alternativen sowie die freie Auswahl von beliebigen Komponenten dargestellt werden. Die derart erstellten Diagramme bilden nun wiederum die Grundlage für die eigentliche Konfiguration des Gesamtsystems, indem der Konfigurator für die Alternativknoten eine passende Komponente auswählt. Die detaillierte Vorgehensweise, wie der Baum zu traversieren und Ersetzungen vorzunehmen sind einschließlich der werkzeugunterstützten Konfiguration ist in Grollius 2010 beschrieben. Insbesondere Tools können bei der Konfiguration helfen, indem sie eine Validierung der entstandenen Gesamtapplikation durchführen können. Auf diese Weise werden Widersprüche, Zyklen oder anderweitige unzulässige Konstellationen und Modellierungsfehler frühzeitig erkannt. Auch bei der Auswahl möglicher Komponenten für den Unendlich-Knoten kann ein passendes Werkzeug durch Anbindung an unterschiedliche Komponenten-Kataloge und -Registries behilflich sein.

Aufgrund der stark komponentenorientierten Ausrichtung von Verbundanwendungen stellt die beschriebene Vorgehensweise auch für Composite Applications eine in-

teressante Konfigurationsmöglichkeit dar. Allerdings beschränkt sich dies, wie eingangs erwähnt, auf die statische Konfiguration: die Gesamtapplikation wird einmalig zusammengestellt und anschließend zur Ausführung gebracht. Um eine andere Komponente zu aktivieren, muss manuell erneut eine Rekonfiguration mit dem Konfigurationswerkzeug vorgenommen werden. Im Gegensatz dazu steht die dynamisch maschinelle Konfiguration, bei der die Varianten aufgrund festgelegter Kriterien automatisch ermittelt und aufgerufen werden. Für manche Geschäftssituationen kann oftmals keine feste Zuordnung vorab vorgenommen werden – sie ergibt sich aufgrund von bestimmten Konstellationen erst zur Laufzeit. So kann es einen gemeinsamen Prozess zur Auftragsabwicklung geben, wobei die aufzurufenden Komponenten allerdings davon abhängig sind, ob es sich um ein Projekt- oder Anlagengeschäft handelt. Für diese Fälle eignet sich die regelbasierte Konfiguration. Die Variante wird dann erst zur Laufzeit ermittelt. Die Kompatibilität der zur Verfügung stehenden Komponenten muss dabei zuvor gewährleistet werden. Hier kann eine Kombination mit der auf VCD basierenden Methode behilflich sein: die durch das Werkzeug als konfliktfrei erkannten und für verschiedene fachliche Konstellationen als sinnvoll ermittelten Varianten werden abgespeichert und über passende Geschäftsregeln zugreifbar gemacht. In den Geschäftsregeln wird hinterlegt, unter welchen Umständen welches zuvor konfigurierte Gesamtsystem aufzurufen ist. Allerdings müssen sämtliche möglichen Gesamtsysteme zuvor statisch festgelegt werden, was aufgrund der möglichen Kombinationen einen nicht unerheblichen Aufwand bedeuten kann. Stattdessen wird die Flexibilität erhöht, indem an allen Stellen des Varianten-Komponenten-Diagramms, an denen sich ein Alternativknoten befindet, ein Regelwerk eingesetzt wird, das, je nach Kontext, die passende Variante ermittelt.

Bei den elementaren Knoten ist sowohl die fachliche als auch die Schnittstellen-Kompatibilität gewährleistet. An unendlich-Knoten können hingegen auch Komponenten eingesetzt werden, die inkompatible Schnittstellen besitzen. In solchen Fällen lassen sich die Diskrepanzen durch Mapping-Funktionalitäten in einem ESB überwinden, solange die fachliche Funktionalität erbracht wird. Wird also der Einsatz schnittstellen-inkompatibler Komponenten erlaubt, erweitert sich das Spektrum potenziell nutzbarer Komponenten deutlich. Ihr Aufruf muss indirekt über eine vermittelnde Middleware realisiert werden, vergleichbar der Servicevertrag-Implementierungsschicht in der Architektur von Verbundanwendungen.

Im Vergleich der Optionen zur Zusammenstellung von dynamischen Anwendungssystemen aus Komponenten bietet das statisch-manuelle Verfahren aufgrund von Verifikationen den unschätzbaren Vorteil, dass daraus eine in sich stimmige und konsistente Gesamtapplikation resultiert. Es ist allerdings zur Laufzeit weniger flexibel und bei Prozessen, die per Definition zur Laufzeit verschiedenen Varianten aufrufen müssen, gänzlich ungeeignet. Hier hat die dynamisch-maschinelle Lösung deutliche Vorteile. Sie kann allerdings keine Konsistenz über die Komponenten hinweg garantieren. Auch hinsichtlich des bereits vielfach diskutierten Begriffs der *losen Kopplung* ist die dynamisch-maschinelle Variante loser gekoppelt als die statisch-manuelle Lösung. Für weitere Details über die flexible Steuerung von Komponenten über Regelwerke sei auf die Diskussion in Kapitel 5.7 verwiesen.

## 4.6 Bedeutung operativer und dispositiver Aspekte bei der Umsetzung von Verbundanwendungen

Ein generelles Problem bei der Umsetzung von Verbundanwendungen ist die Zuweisung geeigneter Ressourcen beim Start der Anwendung. Hier kommt ein weiterer Nachteil der BPMN zum Tragen: die Modelle enthalten keine Details über die Anforderungen an die jeweils benötigte Ressource. Beispielsweise wurden im implementierten Geschäftsprozess dieses Kapitels die Rollen *Antragsteller, Genehmigender* und *Administrator* vergeben. Doch welche Eigenschaften werden von demjenigen erwartet, der die Rolle auszufüllen hat? Welches Wissen und welche Fertigkeiten werden vorausgesetzt? In welchem Verhältnis stehen die Rollen zueinander? Können die Rollen von ein und derselben Person ausgefüllt werden? Sind Vorgabezeiten einzuhalten und wenn ja, kann die Person diese erfüllen?

Ähnlich sieht es bei den verwendeten Services aus: kann jeder Service, der die Schnittstelle implementiert, herangezogen werden? Oder sind gewisse Sicherheits-, Performanz- und/oder Zuverlässigkeitsbedingungen einzuhalten? In welchem Verhältnis steht der aktuell zuzuordnende Service zu bereits zugeordneten Services, müssen also beispielsweise alle Services auf ein und demselben System ablaufen oder kann dies variieren (ein Service *Auftrag aktualisieren* muss notwendigerweise auf demselben System ablaufen wie der zuvor zugeordnete Service *Auftrag anlegen*, während ein eher generischer Service wie *Kreditkarte prüfen* von einem beliebigen Service-Provider erbracht werden kann)?

Derartige Informationen sind in den Modellen in der Regel nicht oder eher informell als Kommentare zu finden, obwohl sie bei der Zuweisung von Ressourcen zum Prozessmodell während der Instanziierung von hoher Bedeutung und somit auch für Verbundanwendungen relevant sind. Es stellt sich also die Frage, wie derartig wichtige Informationen festgehalten und sowohl bei der Modellierung als auch bei der Ressourcen-Zuweisung gewinnbringend (und idealerweise automatisiert) eingesetzt werden können. Diesem Problem widmet sich Link 2010 in seinem Papier über einen zweistufigen Modellierungsansatz zum nachhaltigen Prozessmanagement. Darin wird zum einen eine Unterteilung obiger Anforderungen (in dem Papier im Sinne der aspektorientierten Modellierung mit *Aspekte* bezeichnet) in zwei Aspektebenen (dispositive und operative Aspekte) vorgeschlagen und zum anderen gezeigt, wie diese Aspekte in einer neuen, explizit ausgewiesenen Vorbereitungsphase, die sich zwischen den klassischen Geschäftsprozessphasen Design und Ausführung befindet, vorteilhaft eingesetzt werden können.

Die beiden Aspektebenen veranschaulicht dazu die aus dem Papier stammende Abbildung 63.

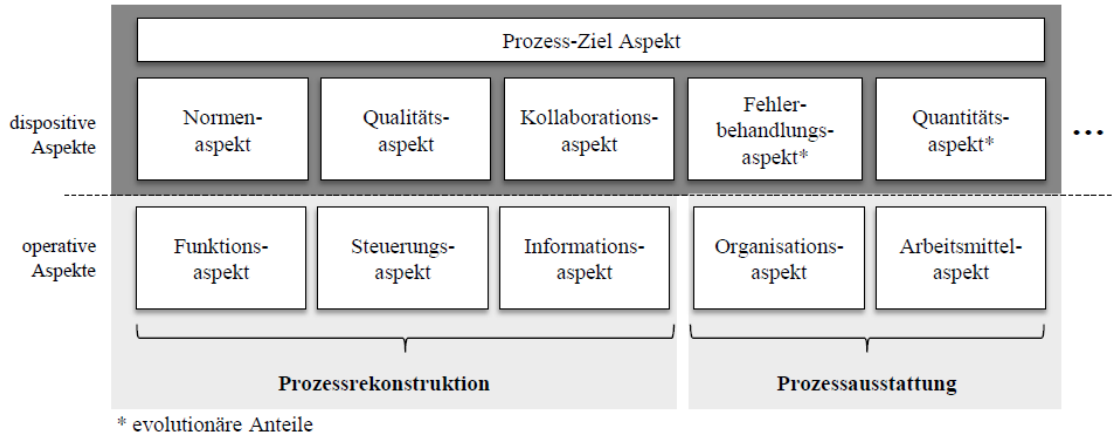



***Abbildung 63: Operative und dispositive Aspekte (aus Link 2010)***

Die operativen Aspekte (in Anlehnung an Jablonski und Bussler 1996, Lehmann 1999 und Jablonski 2008) fassen im Wesentlichen zusammen, was (Funktionsaspekt) von wem (Organisationsaspekt) in welcher Reihenfolge (Steuerungsaspekt) und mit welchen Mitteln (Informations- und/oder Arbeitsmittelaspekt) durchzuführen ist. Die dispositiven Aspekte sind so zu interpretieren, dass sie die darunterliegenden Aspekte der operativen Ebene beeinflussen bzw. einschränken. Unter dem Normenaspekt wer-

den zu berücksichtigende Vorschriften, Regelungen, Richtlinien, Gesetzte, usw. verstanden. Der Qualitätsaspekt hingegen fasst sämtliche qualitätsrelevanten Anforderungen wie der angestrebte Qualitätsstandard zusammen und der Kollaborationsaspekt adressiert demgegenüber die Fragestellung, wer mit wem wie woran zusammenarbeitet. Der Fehlerbehandlungsaspekt berücksichtigt erwartete, allerdings unerwünschte Fehlerereignisse und wie mit ihnen umzugehen ist, während der Quantitätsaspekt alle messbaren Kennzahlen umfasst. Der Prozess-Ziel-Aspekt schließlich priorisiert die Zielsetzungen des betroffenen Prozesses.

Wie bereits eingangs erwähnt, schränken sämtliche Aspekte die Auswahlmöglichkeiten bei der Instanziierung eines Prozesses deutlich ein und tragen somit zur Komplexitätsreduzierung bei. So begrenzt beispielsweise die Vorgabezeit für eine bestimmte Aufgabe die Menge möglicher Kandidaten, oder eine preisliche Obergrenze limitiert zwangsläufig die Menge der einsetzbaren Maschinen. In jedem Fall sorgen Aspekte für ein qualitativ höherwertiges und damit für ein nachhaltigeres Prozessmanagement. So können die operativen Modelle, repräsentiert durch BPMN-Diagramme, jederzeit gegen die Aspekte der dispositiven Ebene validiert werden.

Doch wie kommen die Aspekte nun konkret zum Einsatz? Hierfür sieht Link aufgrund der besonderen Bedeutung der Aspekte eine neue Phase im klassischen Lebenszyklusmodell von Geschäftsprozessen vor. Es handelt sich dabei um eine explizite Vorbereitungsphase, die zwischen der Design- und der Ausführungsphase eingebettet ist. Abbildung 64 veranschaulicht diesen Sachverhalt.

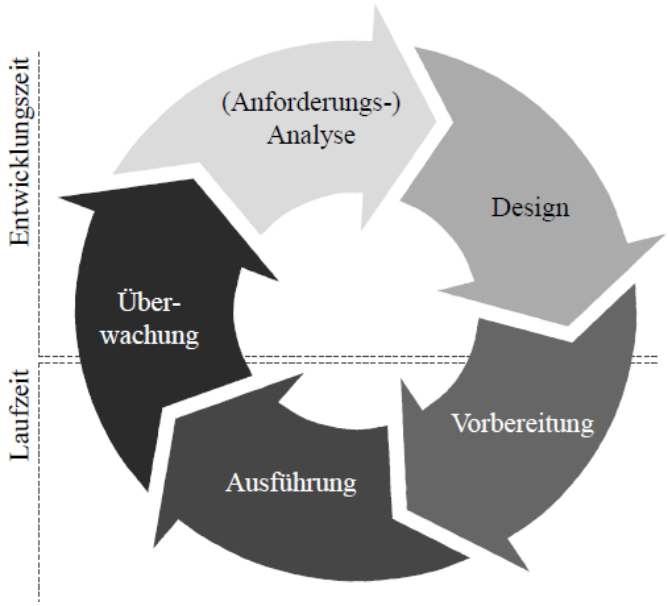



***Abbildung 64: Vorbereitungsphase als neue Phase im Lebenszyklusmodell von Geschäftsprozessen (aus Link 2010)***

Insbesondere vor der eigentlichen Instanziierung des Prozesses müssen die im Modell benötigten Ressourcen konkret zugeordnet werden. In diesem Sinne ist die Ausführung vorzubereiten vergleichbar der Arbeits-Vorbereitungsphase im Fertigungsbereich. Im Zuge dieser Phase findet also genau die Zuweisung der zur Verfügung stehenden Ressourcen konform zu den dispositiven Aspekten statt. Doch sie schließt nicht notwendigerweise mit Beginn der Ausführungsphase ab. Vielmehr ist es aufgrund langer Prozesslaufzeiten wahrscheinlich, dass sich dynamische Ressourcenzuweisungen erst während der Prozessausführung ergeben. Auch diese müssen selbstverständlich den dispositiven Vorgaben folgen. Entsprechend erstreckt sich die Vorbereitungsphase bis zum Laufzeitende des Prozesses. In der nachfolgenden Überwachungsphase kann eine Analyse des beendeten Prozesses gegenüber der dispositiven Aspekte erfolgen. Aufgrund der daraus gewonnenen Erkenntnisse ist eine Anpassung auf dispositiver Ebene denkbar. Dies gilt im Besonderen für die in Abbildung 63 mit „evolutionär“ bezeichneten Aspekte (Fehlerbehandlungsaspekte sowie Quantitätsaspekte). Auf diese Weise findet eine ständige Optimierung der Aspekte statt, was wiederum der Gesamtqualität zugutekommt.

Die von Link vorgeschlagenen Ideen sind auch für Verbundanwendungen eine wertvolle Ergänzung. Die Berücksichtigung einschränkender Bedingungen bei der Ressourcenzuordnung stellt gleichermaßen für Composites eine besondere Herausfor-

derung dar und kann daher durch dispositive Aspekte qualitativ verbessert werden. Dies ist in doppelter Hinsicht nützlich, als dass Verbundanwendungen sowohl betriebswirtschaftliche als auch technische Prozesse adressieren und somit beide Prozessarten von dem Ansatz profitieren könnten. Nichtsdestotrotz muss in weiteren Arbeiten geklärt werden, wie die Aspekte konkret zu formulieren sind, wie Ressourcen mit ähnlichen Informationen versehen werden können und wie anschließend in der Vorbereitungsphase ein Abgleich zwischen geforderten sowie bereitgestellten Eigenschaften zu erfolgen hat. Insbesondere wenn dies automatisiert vonstatten gehen soll, erinnert die Problemstellung sehr an die Suche nach geeigneten Services aufgrund semantischer Informationen wie dies in Kapitel 3.5 diskutiert wurde. Auch wenn der Zuordnungsschritt vorerst nur manuell durchgeführt wird, so stellen die dispositiven Aspekte in jedem Fall eine wertvolle Hilfe dar, um geeignete Mittel optimal zuweisen zu können.

# 5 Ergänzende Konzepte zur Unterstützung der Architektur von Verbundanwendungen

In den beiden vorangegangenen Kapiteln wurde die Grundarchitektur von Verbundanwendungen diskutiert und implementiert. Im weiteren Verlauf der Arbeit wird es jetzt darum gehen, weitere Konzepte zu untersuchen, die im Rahmen von Verbundanwendungen sinnvoll sind und die die beschriebene Architektur zur Erreichung einer besseren Robustheit unterstützen. Es werden dazu folgende Fragestellungen analysiert:

- Locking-Verhalten der angeschlossenen Systeme
- Idempotenz-Anforderung an die aufgerufenen Services
- Ereignisse als Voraussetzung für Rückmeldungen an die wartende Verbundanwendung
- Fehlerbehandlung bei der Kommunikation mit den Backend-Systemen
- Wizard-ähnliche Benutzeroberflächen im Vergleich zur Verwendung von Benutzeroberflächen in Prozessflüssen

Darüber hinaus werden einige Pattern präsentiert, die sich in der Servicevertrag-Implementierungsschicht bewährt haben. Sie veranschaulichen typische Problemstellungen und deren mögliche Lösung. Abgeschlossen wird das Kapitel mit einem Abschnitt über die Steigerung der Flexibilität von Verbundanwendungen durch die Verwendung von Regelwerken in Kombination mit analytischen Anwendungen.

## 5.1 Locking-Verhalten der angeschlossenen Systeme

Verbundanwendungen sind gemäß der in dieser Arbeit beschriebenen Architektur über zustandslose Service-Aufrufe mit den Backend-Systemen lose gekoppelt. Von daher ist stets ungewiss, in welchen zeitlichen Abständen auf die Backend-Systeme zugegriffen wird. Dies spielt insbesondere bei Lese-Schreib-Zyklen eine entscheidende Rolle: da nach einem Lesevorgang nicht vorhersehbar ist, ob auf die Daten überhaupt jemals wieder schreibend zugegriffen wird und wenn ja, zu welchem Zeitpunkt dies erfolgt, scheiden pessimistische Lockingverfahren, bei denen die Daten nach einem Lesevorgang für die Bearbeitungsdauer des Anwenders gesperrt bleiben, von

vornherein aus. Da die Backend-Systeme der Verbundanwendung nicht exklusiv zur Verfügung stehen, sondern mit anderen Anwendungen um sie konkurrieren, führt ein pessimistisches Locking durch eine Verbundanwendung zwangsläufig zu nicht akzeptablen Störungen in anderen Anwendungen, die ebenfalls Zugriff auf die Daten benötigen.

Von daher müssen lesende Zugriffe über Services mit optimistischem Locking durchgeführt werden. Ob ein Backend-System optimistisches Locking bei seinen Service-Aufrufen unterstützt, lässt sich anhand der Signatur der Services erkennen. Enthalten die zu übergebenden bzw. die zurückgelieferten Daten sowohl beim lesenden als auch bei den schreibenden Aufrufen einen Zeitstempel, wann sie zum letzten mal verändert wurden oder zumindest eine Sequenznummer, die bei jeder Aktualisierung der Daten durch ein Inkrement verändert wird, so wird optimistisches Locking unterstützt. Das Backend-System wird bei einer nachfolgenden Schreiboperation den dann mitgelieferten Zeitstempel bzw. die dann mitgelieferte Sequenznummer mit dem aktuellen Wert in der Datenbank vergleichen und bei unterschiedlichen Werten durch eine Ausnahme einen Schreibkonflikt anzeigen. Auf diese kann anschließend entsprechend reagiert werden.

Sieht das Backend-System hingegen eine solche Verarbeitung nicht vor, so muss die Funktionalität in der Servicevertrag-Implementierungsschicht nachgebildet werden. Dies könnte beispielsweise dadurch erfolgen, dass die Daten zum Lesezeitpunkt auch in die Implementierungsschicht kopiert werden. Während des Schreibvorgangs werden die Daten vor der eigentlichen Verbuchung durch die Implementierungsschicht nochmals aus dem Backend gelesen, dabei gleichzeitig im Backend gesperrt, und mit der Kopie verglichen. Stimmen die nachgelesenen Daten mit denen der Kopie überein, hat zwischenzeitlich kein weiterer Schreibvorgang stattgefunden und die Daten können geschrieben werden. Andernfalls liegt ein Schreibkonflikt vor, der dann wiederum durch eine Ausnahme angezeigt wird. Die gelockten Daten sind in diesem Fall wieder zu entsperren.

Die bisher beschriebenen Verfahren haben allerdings einen gravierenden Nachteil: Lese- und Schreiboperationen müssen zusammenarbeiten, also als gut aufeinander abgestimmte Pärchen implementiert werden. Beim Lesen muss entweder das gesamte Objekt (wenn das Backend kein optimistisches Locking unterstützt) bzw. der Zeit-

stempel/die Sequenznummer (wenn das Backend optimistisches Locking unterstützt) in der Servicevertrag-Implementierungsschicht in einer separaten Tabelle hinterlegt werden, auf die dann zum Schreibzeitpunkt wieder zuzugreifen ist, um die Funktionalität der Konflikterkennung umsetzen zu können. Erschwerend kommt die Zuordnung der Daten zum lesenden Client hinzu: welche Benutzeroberfläche oder welcher Prozess hat die Daten gelesen? Also ist ergänzend entweder die Session-ID des Clients oder die Prozess-ID des betroffenen Prozesses zusammen mit den Daten in der Tabelle abzuspeichern. Was passiert aber dann, wenn die Benutzeroberfläche oder der Prozess beendet werden, ohne dass eine weitere Schreiboperation folgt? Wer bereinigt dann die Tabelle, die ja nur während der Lebensdauer des UIs bzw. des Prozesses sinnvoll ist? Natürlich könnten regelmäßige Batch-Jobs überprüfen, ob Prozesse noch laufen und ggf. die Tabelle bereinigen, wenn dem nicht so wäre. Auch die Einträge für die mitprotokollierten aktiven Clients könnten entfernt werden, wenn über eine gewisse Zeitspanne keine Reaktion mehr erfolgte. Das setzt allerdings wieder voraus, dass man bei Benutzeroberflächen auch den Zeitpunkt der letzten Kommunikation mitprotokollieren müsste. Der Verwaltungsaufwand und damit die Fehleranfälligkeit steigen also dadurch beträchtlich. Folglich ist zu überlegen, ob es nicht doch eine einfachere Alternative gibt.

In der Tat lässt sich eine einfachere Abwicklung finden, die darauf beruht, dass von der Verbundanwendung beim Schreiben neben den neuen Daten auch die ursprünglichen, unveränderten Daten (also so, wie sie zum Lesezeitpunkt vorlagen) als Parameter mit übergeben werden. Vor der eigentlichen Schreiboperation in das Backend-System (die Backendsysteme) muss dann zwar noch eine Leseoperation und ein Vergleich der von dem Backend gelesenen Daten mit den von der Composite übergebenen alten Daten durchgeführt werden, um die Konflikterkennung sicherzustellen. Allerdings ist bei diesem Vorgehen keinerlei zusätzliche Verwaltung in der Servicevertrag-Implementierungsschicht einzurichten, da sämtliche Informationen zur eindeutigen Konflikterkennung durch die Parameterübergabe von der Verbundanwendung an die Schreiboperation vorliegen. Die Leseoperation muss also keinerlei administrativen Daten sammeln, auf die eine spätere Schreiboperation dann angewiesen wäre.

Außerdem hat dieses Verfahren einen weiteren Vorteil: der Vergleich erfolgt nur auf die wirklich änderungsrelevanten Felder eines Objekts. Mit "änderungsrelevant"

sind in diesem Zusammenhang die Felder gemeint, die die Verbundanwendung aktuell modifizieren möchte.

Ein Beispiel soll diesen Sachverhalt verdeutlichen: in der Composite wird die Lieferadresse zu einem Auftrag von "Köln" auf "Bonn" verändert. Beim Schreiben werden beide Orte geliefert (vorher/nachher) und vor der eigentlichen Aktualisierung wird nachgeprüft, ob "Köln" auch noch im ursprünglichen Auftrag im Backend vorliegt. Ist dem so, kann die Aktualisierung vorgenommen werden, andernfalls liegt ein Konflikt vor. Wurde an dem Auftrag parallel von einer anderen Anwendung die Straße der Rechnungsadresse geändert, so bleibt der Schreibvorgang für den Ort der Lieferadresse davon unberührt, da in diesem Fall nur dieser Ort als das änderungsrelevante Feld anzusehen ist.

Bei einem Verfahren, dass nur das Objekt "Auftrag" als Einheit optimistisch sperren kann, wäre dies nicht möglich gewesen. Denn dann hätte die Veränderung der Straße in der Rechnungsadresse eine Veränderung des Zeitstempels bzw. der Sequenznummer für den gesamten Auftrag zur Folge gehabt. Beim anschließenden Schreibvorgang der Lieferadressenänderung, die sich ja ebenfalls auf den Auftrag bezieht, kommt es dann zwangsläufig zu einem Konflikt. In Summe ist das beschriebene Verfahren also einfacher und dabei weniger konfliktträchtig als die Implementierung zusätzlicher Verwaltungstabellen beim Zusammenspiel zwischen Lese- und Schreiboperationen. Der Nachteil liegt in einer zusätzliche Leseoperation und dem Vergleich der Objektzustände in Verbundanwendung und Backendsystem. Da allerdings die Schreiboperationen aufgrund der lose gekoppelten Architektur ohnehin asynchron abgewickelt werden, sollte der damit verbundene zeitliche Verzug verschmerzbar sein.

Das Verfahren des optimistischen Lockings ist bei langlaufenden Transaktionen bei gleichzeitig relativ geringem Wettbewerb um dieselben Daten sinnvoll. Schließlich muss jedem Konflikt nachgegangen werden. So könnte ein Konflikt durch eine Fachabteilung gelöst werden, in dem die Daten aus der Datenbank mit dem neuen Datensatz abgeglichen und gemäß betriebswirtschaftlichen Vorgaben geeignet zusammengeführt werden. Teilweise ließen sich solche Merge-Vorgänge auch automatisieren, so dass nur ein kleiner Teil durch Interaktion mit Fachexperten gelöst werden muss. Alternativ kann das betroffene Geschäftsobjekt in feingranularere Teilobjekte unterteilt werden, die die Konfliktwahrscheinlichkeit sinken lassen. Eine weitere Möglichkeit

zur Verringerung der Wahrscheinlichkeit eines Konflikts ist die Verwendung sogenannter Check- oder Simulate-Services, wie sie beispielsweise in SAP-Systemen üblich sind. Diese sind ebenfalls von den Backend-Systemen zur Verfügung zu stellen und ermöglichen eine Simulation der auszuführenden Schreiboperation ohne dabei jedoch die Daten im Backend zu verbuchen. Es werden lediglich die Voraussetzungen für eine spätere erfolgreiche Verbuchung geprüft. Bei modernen Systemen, die von der Möglichkeit einer Teilnahme an lose gekoppelten Szenarien im SOA-Umfeld ausgehen, sind derartige Services durchaus keine Seltenheit. Sie werden aus dem UI der Verbundanwendung heraus über die Servicevertrag-Implementierungsschicht aufgerufen. Zwar können auch sie letztendlich keine vollständige Sicherheit für die Vermeidung eines späteren Schreibkonflikts geben, da die Daten ebenfalls nicht gesperrt werden. Da dem Simulate-Aufruf allerdings unmittelbar danach der eigentliche Verbuchungsaufruf folgen sollte (ohne dass dazwischen noch eine zeitaufwändige Benutzer-Interaktion stattfinden darf), wird durch die geringe Zeitspanne, die zwischen den beiden Aufrufen liegt, die Wahrscheinlichkeit eines Konflikts deutlich gesenkt.

Nicht zu empfehlen ist sicherlich ein expliziter Check-In-/Check-Out-Mechanismus auf Anwendungsebene. Diese Lösung ist schon deshalb untauglich, da existierende Anwendungen, die ebenfalls auf den Daten arbeiten, auch dieses Vorgehen implementieren müssten. Aufgrund der Nicht-Invasivitätsanforderung von Verbundanwendungen scheidet diese Alternative folglich aus.

## 5.2 Idempotenz

Die Idempotenz-Anforderung von Serviceaufrufen ist sicherlich eine der Kerneigenschaften in lose gekoppelten Architekturen und muss wiederum von den Backend-Systemen umgesetzt werden. Hohpe & Woolf (2004) beschreiben dazu in ihrem Buch *Enterprise Integration Patterns* das *Idempotent Receiver*-Pattern. Im Wesentlichen geht es bei der Idempotenz darum zu erkennen, dass ein und dieselben Serviceaufrufe aufgrund von technischen Störungen mehrfach auftreten, und entsprechende Vorsichtsmaßnahmen zu ergreifen, die fachliche Probleme vermeiden. Ein klassisches Beispiel ist die Bestellung eines teuren Wirtschaftsgutes: nach dem Versand der Bestellung erfolgt keine Quittierung seitens des Lieferanten, so dass der Besteller seinen Auftrag erneut abschickt. Der Besteller weiß allerdings nicht, ob sein Aufruf auf dem Weg *zum Lieferanten* oder die Quittierung auf dem Weg *zum Besteller* gestört wurde.

Also kann es passieren, dass der Besteller das Wirtschaftsgut aufgrund einer Doppelbestellung zweimal geliefert bekommt, nämlich dann, wenn die Bestellung bereits beim ersten Versuch erfolgreich beim Lieferanten verbucht werden konnte und erst die Quittung aufgrund einer Störung verloren geht.

Zur Vermeidung dieses Problems hat sich ein relativ einfacher Algorithmus etabliert. Wiederum wird die Service-Schnittstelle um einen zusätzlichen Parameter erweitert. Dieser Parameter enthält eine eindeutige Nummer, wie sie in modernen Programmiersprachen von Bibliotheksfunktionen generiert werden kann (z.B. in Java durch die Funktion `java.util.UUID.randomUUID()`). Der Service-Aufrufer wird eine einmal generierte Nummer lediglich im Wiederholungsfalle (Retry) wiederverwenden. Ansonsten generiert er für jeden Aufruf einen neuen Identifikator. Die Service-Implementierung pflegt nun wiederum eine Tabelle mit Identifikationsnummern. Bevor die Daten verbucht werden, erlaubt eine Suche in dieser Tabelle nach der übergebenen ID die Identifikation eines Duplikats. Befindet sich die Nummer noch nicht in der Tabelle (es handelt sich also um den ersten Aufruf), so werden die Daten in dem Backend-System verbucht, die Antwortnachricht erstellt und an den Aufrufer zurückgeschickt. Vor dem eigentlichen Versand der Daten wird allerdings noch die übergebene Identifikationsnummer zusammen mit den zurückzusendenden Daten in der Tabelle hinterlegt. Bei einem erneuten Aufruf mit den selben Daten kann jetzt aufgrund des Tabelleneintrags das Duplikat erkannt werden. In diesem Fall müssen die Daten nicht im Backend verbucht werden. Stattdessen wird nochmals die zur ID abgespeicherte Nachricht an den Aufrufer gesendet.

Ob ein Backend-System diese Funktionalität zur Verfügung stellt, lässt sich folglich wiederum an der Signatur des Services erkennen. Auch hier ist die Servicevertrag-Implementierungsschicht gefordert, wenn Idempotenz nicht unterstützt wird. Der Aufwand zur Implementierung hält sich allerdings in Grenzen, wie die Beschreibung des Algorithmus gezeigt hat.

## 5.3 Ereignisse

Im Kapitel über die Grundarchitektur von Verbundanwendungen spielt bei asynchronen Schreiboperationen die Benachrichtigung des wartenden fachlichen Prozesses durch den technischen Prozess in Form eines Ereignisses, auf das der fachliche Prozess wartet, eine entscheidende Rolle. Natürlich muss der Fachprozess nicht generell warten. Ist er auf das Ergebnis des Serviceaufrufs im weiteren Prozessablauf nicht angewiesen, ist das Warten überflüssig. Stattdessen kann er normal im Prozess fortfahren.

Ist hingegen die erfolgreiche Verbuchung im Backend für den weiteren Prozessverlauf essenziell, so muss seitens der Servicevertrag-Implementierungsschicht ein entsprechender Mechanismus über Eventing zur Verfügung gestellt werden, damit der Fachprozess aus seinem Wartezustand befreit werden kann. Im BPMN-Diagramm wurde dies mittels des nachrichtenbasierten Zwischenereignisses im Fachprozess dargestellt.

Seitens der Servicevertrag-Implementierungsschicht sind nun drei Fälle zu unterscheiden, wie die erfolgreiche Verbuchung im Backend festgestellt und dem Fachprozess mitgeteilt werden kann:

- Das Backend-System wird synchron angesprochen: in diesem Fall wird der Erfolg anhand der Rückgabeparameter des synchronen Aufrufes erkannt. Die Rückgaben können gemäß der Anforderungen des fachlichen Prozesses zu einem Event zusammengefasst und an die Composite gesendet werden.

- Das Backend-System wird asynchron angesprochen und das Backend versendet zeitversetzt ein entsprechendes Acknowledgement in Form eines Ereignisses: in diesem Fall ist das Backend-System selbst in der Lage, Ereignisse auszulösen, auf die der technische Prozess lediglich zu warten hat. Es handelt sich aus dem Blickwinkel des technischen Prozesses also um ein klassisches Request-Reply-Pattern (Hohpe & Woolf 2004), auf das noch im Pattern-Kapitel 5.6 einzugehen sein wird. Der technische Prozess transformiert nach Erhalt des Ereignisses dessen Inhalt lediglich in das vom fachlichen Prozess erwartete Format und verschickt es.

- Das Backend-System wird asynchron angesprochen, es verschickt aber selbstständig keinerlei Quittungen: zur Behandlung dieses Falles muss der technische Prozess selbst aktiv werden und das Backend-System in regelmäßigen Abständen pollen, bis anhand eines Zustandskennzeichens in dem durch die Schreiboperation betroffenen Geschäftsobjekt der Erfolg oder Misserfolg erkannt werden kann. Abbildung 65 zeigt eine Möglichkeit, wie Polling in BPMN umgesetzt werden kann.

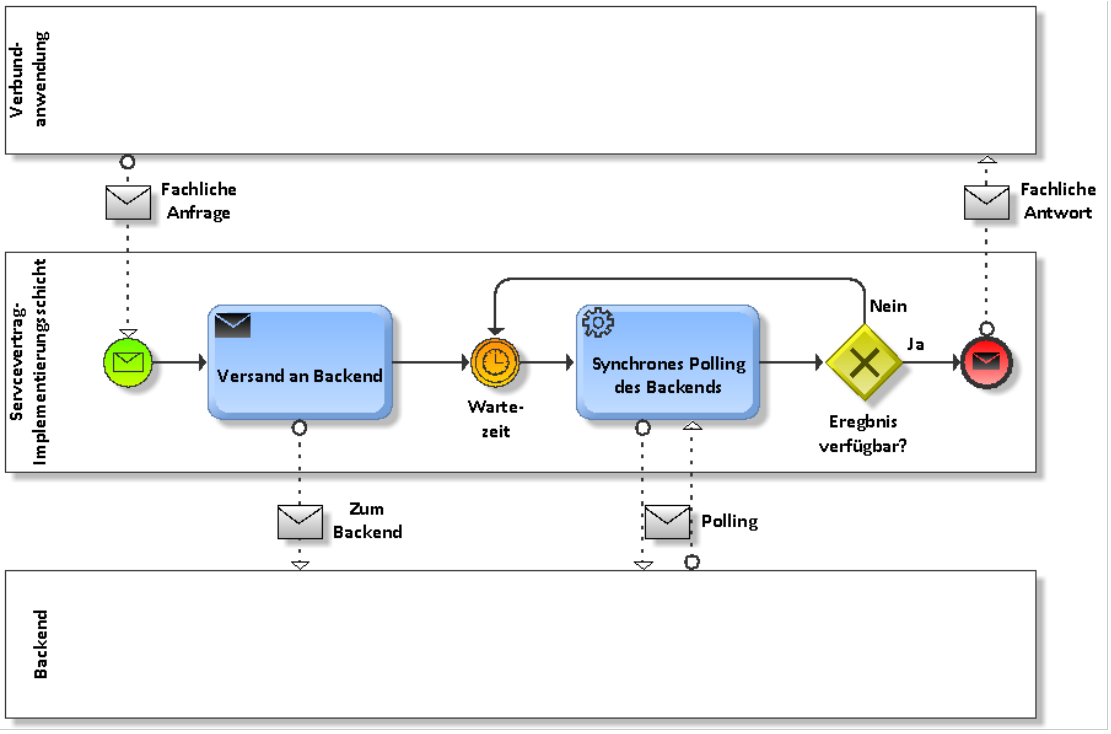



*Abbildung 65: Synchrones Polling*

Das Modell zeigt die Verbundanwendung im oberen Pool, in der Mitte die Servicevertrag-Implementierungsschicht und im unteren Pool das Backend-System. Initiiert wird der Datenaustausch über eine asynchrone Nachricht von der Composite. Der technische Prozess leitet die Anfrage nun an das Backend-System weiter, von dem bekannt ist, dass es selbst keine Reaktion zeigen wird. Daher wird aus dem technischen Prozess heraus nach einer gewissen Wartezeit ein Status synchron vom Backend-System abgefragt. Ist das gewünschte Ergebnis, egal ob positiv oder negativ, eingetreten, wird ein entsprechendes Ereignis an den wartenden Fachprozess gefeuert. Ist die Backend-Verarbeitung hingegen noch nicht abgeschlossen, dass erwartete Ergebnis also nicht einge-

troffen, so wird nach einer vordefinierten Wartezeit das Backend-System erneut angefragt. Dieser gesamte Prozess ließe sich jetzt wieder mit den bereits diskutierten BPMN-Mitteln zeitlich überwachen, worauf aus Gründen der Übersichtlichkeit bewusst verzichtet wurde.

Wie in BPMN üblich gibt es auch für dieses Beispiel mehrere Implementierungsmöglichkeiten. Alternativ zum gezeigten Modell lässt sich die Schleife auch durch einen Unterprozess mit Schleifenbedingung abbilden, wie dies in Abbildung 66 dargestellt ist.

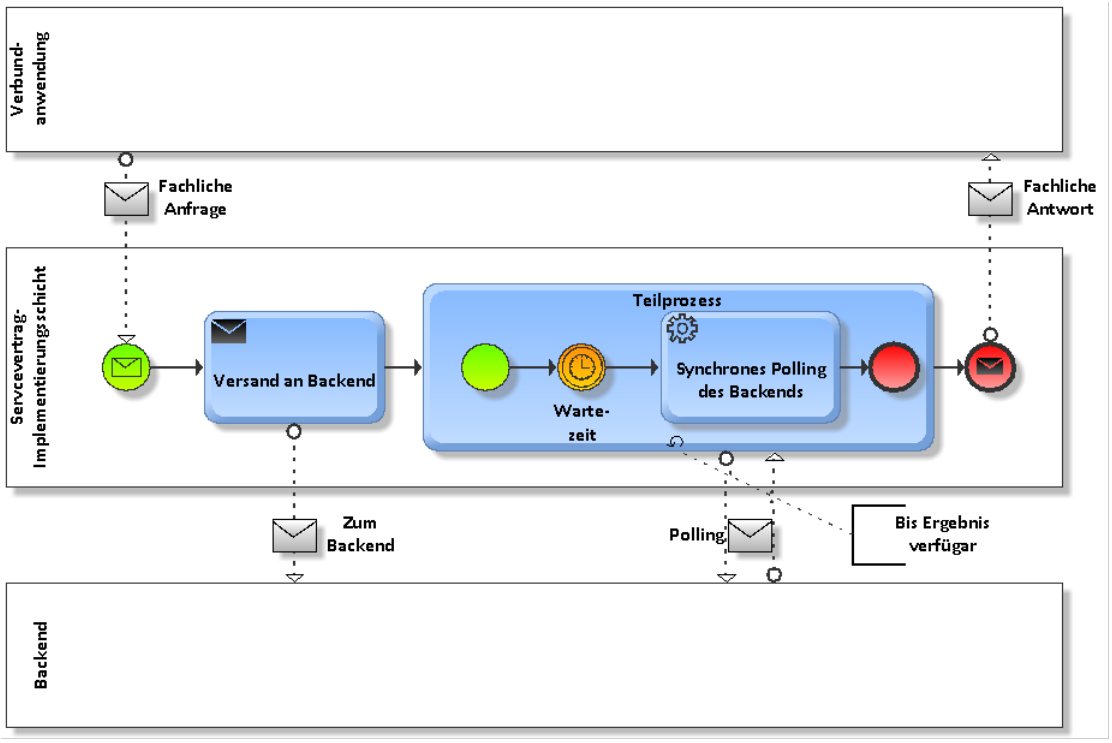



*Abbildung 66: Polling über einen Unterprozess mit Schleifenbedingung*

Der in der Abbildung dargestellte Teilprozess wird nun sooft wiederholt, bis die Schleifenabbruchbedingung erfüllt ist. Da die Anzahl der Schleifendurchläufe im voraus unbekannt ist, wird in BPMN dem Teilprozess das Symbol für den Standard-Schleifendurchlauf hinzugefügt, um diesem Sachverhalt Rechnung zu tragen.

In beiden Prozessmodellen wurde das Backend-System synchron gepollt. Die Umsetzung eines asynchronen Pollings ist schließlich in Abbildung 67 visuali-

siert.

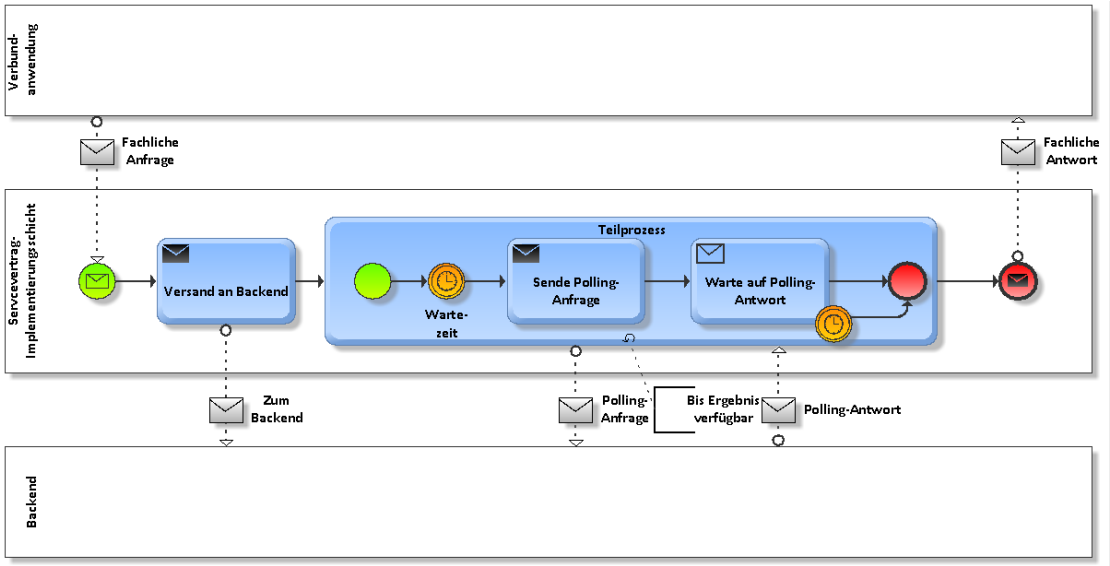



*Abbildung 67: Asynchrones Polling*

Zusätzlich ist in diesem Modell auch das Warten auf die Antwort zeitlich überwacht, um Deadlock-Situationen zu vermeiden. Dies wird in BPMN durch das angeheftete Zeitereignis an der Warte-Aufgabe umgesetzt.

Je nach Eigenschaften der zu integrierenden Backend-Systeme stehen der Servicevertrag-Implementierungsschicht folglich unterschiedliche Mechanismen zur Erzeugung der notwendigen Ereignisse zur Verfügung, um den Fachprozess aus dem Wartezustand zu befreien. BPMN ermöglicht auch in diesen Fällen eine geeignete Umsetzung.

## 5.4 Fehlerbehandlung

Im Rahmen der Diskussion über die Bedeutung der BPMN für die Implementierung von Verbundanwendungen (Kapitel 4.1) wurden bereits einige Prozessmodelle vorgestellt, wie unterschiedlichste Fehlerfälle mittels geeigneter BPMN-Konstrukte abgedeckt werden können (Abschnitt 4.1.3: Ausnahmebehandlung). In diesem Abschnitt geht es nun darum, die unterschiedlichsten Fehlersituationen für Verbundanwendungen zu erkennen und geeignete Maßnahmen zu deren Behebung bzw. Linderung zu ergreifen.

Im Vergleich zu herkömmlichen, eng gekoppelten Anwendungen, erhöht sich die Zahl möglicher Fehlerursachen in lose gekoppelten Szenarien deutlich. Allein die Anzahl der beteiligten Komponenten (Software, Hardware, Leitungen) und der damit verbundenen Möglichkeit eines Ausfalls lässt erahnen, welches Gefahrenpotenzial es zu berücksichtigen gilt. Grundsätzlich kann bei der Fehlerkategorisierung zwischen technischen und fachlichen Fehlern unterschieden werden. Technische Fehler sind nicht vorhersehbar und können vom Endbenutzer einer Anwendung nicht behoben werden. Zur Behebung werden in der Regel Spezialisten wie System-Administratoren, Experten für spezielle Hardware oder Software-Entwickler benötigt. Zu dieser Kategorie zählen beispielsweise Softwarefehler (Bugs), Datenbankabstürze, Routerausfälle oder Leitungsprobleme. Sie fallen eher selten an (bzw. sollten selten auftreten) und sollten so schnell wie möglich entdeckt und behoben werden, da in der Regel eine Vielzahl von Anwendern davon betroffen ist.

Fachliche Fehler/Ausnahmen sind vorhersehbar und sind demnach Bestandteil der normalen Geschäftslogik. Folglich treten sie auch deutlich öfter auf und müssen schon bei der Spezifikation einer Anwendung explizit berücksichtigt werden, beispielsweise durch Validierungslogik in der Benutzeroberfläche und entsprechenden Fehlermeldungen bei deren Verletzung oder auf Prozessebene durch Nutzung entsprechender BPMN-Konstrukte, wie dies in Abschnitt 4.1.3 gezeigt wurde. Für diese Fehlerkategorie gibt es unzählige Beispiele, da sie zum normalen Geschäftsalltag gehören. Anwender vertippen sich, geben falsche Produktnummern ein, die Relation zwischen Eingabefeldern stimmt nicht (eine Ende-Datum liegt vor dem Start-Datum), ein Produkt ist nicht mehr verfügbar usw. Gerade das letzte Beispiel deutet bereits darauf hin, dass auch Schreibkonflikte in lose gekoppelten Szenarien mit in diese Kategorie fallen: ihr Eintreten ist vorhersehbar, also muss eine Konfliktlösungsstrategie vorgesehen werden.

Es stellt sich nun zwangsläufig die Frage, wie die beiden Fehlerursachen in Verbundanwendungen einschließlich der Servicevertrag-Implementierungsschicht zu behandeln sind. Dazu sei zunächst die Verbundanwendung selbst betrachtet. Die Fehlerbehandlung gestaltet sich hier relativ einfach: fachliche Fehler können dem Endanwender unmittelbar in der Benutzeroberfläche angezeigt werden. Er ist auch derjenige, der aufgrund seines fachlichen Wissens diese Fehler am besten lösen kann. Unterstützt wird er dabei von der Software durch entsprechende Validierungslogik. Da aus

dem UI heraus nur synchrone Aufrufe erfolgen sollten, werden auch technische Probleme zumindest sofort erkannt. Sie sollten allerdings für den Endanwender in benutzerfreundlichen, aussagekräftigen Meldungen aufbereitet werden. Ihm direkt einen Java-Stacktrace anzuzeigen, macht sicherlich wenig Sinn. Idealerweise wird in solchen Fällen im Hintergrund automatisch das Servicepersonal informiert, um eine schnellstmögliche Behandlung des Problems zu gewährleisten.

In asynchronen Szenarien wird die Fehlerbehandlung zwischen Verbundanwendung und Servicevertrag-Implementierungsschicht aufgeteilt. Der fachliche Prozess übergibt die während seines Prozessablaufs angesammelten Daten gemäß Servicevertrag asynchron an den technischen Prozess. Ab diesem Zeitpunkt ist der technische Prozess für die weitere Fehlerbehandlung zuständig. Nur die im Servicevertrag vereinbarten fachlichen Fehlermeldungen werden an den wartenden Prozess zurückgeliefert. Dieser ist per Definition auf den Fehler vorbereitet und kann ihn mit einem passenden Sequenzfluss im BPMN-Modell adressieren und explizit fachlich korrekt behandeln.

Technische Fehler müssen hingegen komplett in der Servicevertrag-Implementierungsschicht abgewickelt werden, da eine Verbundanwendung seine angeschlossenen Systeme nicht kennt. Verbundanwendung und Backend-Systeme sind durch die asynchrone Kommunikation ohnehin voneinander getrennt, so dass der Initiator der Kommunikation ohnehin nicht mehr zeitnah konsultiert werden kann.
Stattdessen stehen dem technischen Prozess nun wiederum verschieden Fehlerbehandlungsstrategien zur Verfügung (Peter 2009). So können temporäre Konflikte wie beispielsweise ein gesperrtes Objekt durch einen automatischen Wiederholungsmechanismus adressiert werden: nach einer gewissen Verzögerung wird der fehlgeschlagene Aufruf erneut versucht. Dabei können Wiederholungshäufigkeit und Zeitverzögerung idealerweise konfiguriert werden. So ließe sich festlegen, ob der Aufruf nach einer bestimmten, konstanten Zeitspanne wiederholt werden soll oder ob sich die Wartezeit nach jedem Fehlversuch gemäß eines Algorithmus' automatisch verlängert. Außerdem kann nach Erreichen der vorher festgelegten Anzahl von Wiederholungen eine Eskalation initiiert werden, die das Hinzuziehen eines Experten (sowohl fachlich als auch technisch, je nach Fehlerursache) vorsieht. Scheitert also die automatische Fehlerauflösung, steht als nächste Stufe die manuelle Korrektur an. In diesem Fall kann ein Fachexperte als Stellvertreter für den Endanwender der Verbundanwendung Korrekturen

an den Daten vornehmen und so für eine ordnungsgemäße Weiterverarbeitung sorgen. Oftmals stellen sich aber auch fehlerhafte Einstellungen in den Backend-Systemen selbst als Ursache heraus. Auch hier kann der Experte unterstützend eingreifen. In beiden Fällen kann anschließend durch einen Neustart der ursprünglich abgewiesenen Nachricht die Prozessfortsetzung gewährleistet werden.

Ist ein Anstarten der Nachricht hingegen nicht mehr möglich, so können die Daten notfalls auch manuell in dem Backend-System erfasst werden. Auch hierzu bedarf es eines Fachexperten, der die betriebswirtschaftlichen Abläufe des betroffenen Backends kennt und aufgrund dieses Wissens die passenden Transaktionen durchführen kann. Gegebenenfalls muss er dazu auch Rücksprache mit dem Endanwender halten, damit dessen Auftrag schließlich fehlerfrei verbucht werden kann.

Moderne ERP-Systeme unterstützen die Fehlerbehandlung in lose gekoppelten Szenarien mittlerweile durch technische Frameworks, die sich auf die Fehlerbehandlung von fehlgeschlagenen Service-Aufrufen spezialisiert haben. Unter anderem hat sich in SAP-Backend-Systemen das sogenannte Forward-Error-Handling (FEH) etabliert, dessen Funktionsweise im Folgenden kurz skizziert wird (siehe SAP 2010d und SAP 2010e). Das Ziel dieses Frameworks ist die Vermeidung doppelter Aufwände bei der Behandlung von Fehlern während der Implementierung von Diensten. Vergleichbar der Cross-Cutting Concerns in der aspektorientierten Programmierung wird die eigentliche Fehlerbehandlung an das Framework delegiert und damit aus der eigentlichen Funktionsimplementierung des Dienstes entfernt. Dazu definiert jeder Service lediglich, welche Fehler und Konflikte er anzeigt. In einer Administrationskonsole des Frameworks kann jedem Fehler/Konflikt seine individuelle Auflösungsstrategie zugewiesen werden. Dazu zählen beispielsweise die einleitend erwähnten Wiederholungsperioden, Verzögerungsstrategien sowie Eskalationspfade. Folglich umfasst ein solches Framework auch eine Workflow-Umgebung, um die Einbeziehung von sowohl fachlichen als auch technischen Experten zu gewährleisten. Ist zudem zwischen Service-Implementierung und Fehlerbehandlungs-Framework auch noch eine Callback-Funktionalität implementiert, kann der fehlersignalisierende Dienst während der Fehlerauflösung auch noch vom Framework zurückgerufen werden, um diesem die Gelegenheit zu geben, abschließende Aufräumarbeiten wie das Schließen von Dateien oder Datenbankverbindungen zu verrichten.

Weitere Funktionalitäten eines solchen FEH-Frameworks sind die Gruppierung und Kategorisierung von Fehlern, um ihnen eine einheitliche Bearbeitungsstrategie zuweisen zu können. Dadurch wird die Konfiguration vereinfacht, da nicht jedem einzelnen Fehler eine separate Behandlung vorgegeben werden muss. Stattdessen werden gleichartige Fehler zu einer Gruppe zusammengefasst und einer Kategorie zugeordnet. Der Kategorie wiederum wird dann wiederum eine Fehlerbehandlungsstrategie zugewiesen. Gerade bei einer Vielzahl feingranularer Fehlerursachen macht sich diese Vorgehensweise bezahlt, da sie den Verwaltungsoverhead doch deutlich reduziert.

Zur Laufzeit werden die eingetretenen Fehlerfälle in einer Überwachungskonsole gelistet. Je nach Ursache kann der Fehler gemäß der bereits beschriebenen Strategien behandelt werden. Das Framework stellt in diesem Zusammenhang die zeitnahe Bearbeitung durch Eskalationspfade und entsprechenden Benachrichtigungsmeldungen sicher und regelt zudem die Kommunikation mit den betroffenen Diensten über die vereinbarte Callback-Schnittstelle. Durch die Überwachung der Fehlerbehandlung mittels eines zentralen Frameworks kann auch jederzeit nachgewiesen werden, welchen Weg eine Nachricht durch diese Infrastruktur genommen hat. Dies spielt insbesondere bei den immer wichtiger werdenden Compliance-Anforderungen heutiger Unternehmen eine entscheidende Rolle, da die Nachvollziehbarkeit der Fehlerbearbeitung gewährleistet ist.

## 5.5 Wizard-UIs vs. UI-Verwendung in Prozessschritten

Ein immer wiederkehrendes Problem bei der Implementierung von Verbundanwendung ist das der Entwicklung von Benutzeroberflächen (User Interfaces oder kurz UIs). Innerhalb einer Verbundanwendung kann eine Benutzeroberfläche entweder zum Start eines Prozesses oder während der Prozessausführung als Implementierung eines Prozessschrittes Verwendung finden. Dabei sind die technischen Anforderungen an Benutzeroberflächen für Verbundanwendungen deutlich anspruchsvoller als die von klassischen Anwendungen, bei denen lediglich gegen eine der Anwendung dediziert zugeordnete Datenbank programmiert wird. Im Gegensatz dazu muss eine Oberfläche für eine Verbundanwendung bei dem Abruf externer Dienste stets der Heterogenität der verwendeten Systemlandschaft als auch der damit verbundenen Probleme wie Transaktionalität, erhöhte Ausfallwahrscheinlichkeit der beteiligten Systeme sowie verlängertes Antwortzeitverhalten Rechnung tragen. Von daher wurde im Rahmen

dieser Arbeit empfohlen, die synchronen Aufrufe lediglich aus den Benutzeroberflächen heraus abzuwickeln und die Schreiboperationen in einem nachgelagerten Prozessschritt asynchron an die Servicevertrag-Implementierungsschicht zu übergeben. Darüber hinaus werden Aufgaben in Verbundanwendungen in der Regel von verschiedenen Rollen erfüllt, d.h. Aufgaben werden über Auftragslisten an die Prozessbeteiligten verteilt. Aus der Auftragsliste heraus navigiert der Endanwender zu dem UI, das dem jeweiligen Prozessschritt zugeordnet ist. Diese Arbeitsweise unterscheidet sich wiederum fundamental von herkömmlichen, datengetriebenen Anwendungen, bei denen Endanwender aktiv konkrete Transaktionen und damit Benutzeroberflächen aufrufen.

Für Verbundanwendungen stellt sich nun die Frage, wie feingranular die Oberflächen eines Prozessschritts sein sollten. Es zeigt sich, dass die Benutzeroberfläche für eine Verbundanwendung grundsätzlich rollenorientiert ausgerichtet sein muss und den maximal möglichen Funktionsumfang abdecken sollte, der zum Zeitpunkt der Ausführung möglich ist. Im Umkehrschluss verlangt diese Anforderung den Verzicht auf feingranulare Oberflächen, die über zwei Prozessschritte für ein und dieselbe Rolle miteinander verbunden sind. Die Modellierung von zwei unmittelbar aufeinanderfolgender BPMN-Benutzeraufgaben für ein und dieselbe Rolle macht von daher keinen Sinn. Im Gegenteil: ein derart modellierter Prozess verlangt für die betroffene Rolle den Umweg über die Auftragsliste, um zum nächsten Prozessschritt zu gelangen. Dieses Verhalten wird als störend empfunden. Damit komplexe Oberflächen trotzdem handhabbar bleiben, empfiehlt sich die Verwendung sogenannter Wizards oder der Einsatz von Laschen. Bei einem Wizard wird die Oberfläche in zusammengehörige Einzelmasken aufgeteilt, die nacheinander vom Endanwender abzuarbeiten sind, wobei die Maskenabfolge selbst in der Oberfläche festgelegt wird. In der Beispielimplementierung aus Abschnitt 4.2.4 wurde ein solcher Wizard zur Eingabe der Bestellung und der lokalen Abspeicherung verwendet. Die Oberfläche erhält auf diese Weise einen prozessähnlichen Charakter, ohne auf einen schwergewichtigen Prozess zurückgreifen zu müssen. Die Vorgehensweise erlaubt dem Entwickler zudem, durchaus komplexe Logik strukturiert und damit leicht handhabbar zu gestalten. Durch das Zurücknavigieren entlang der Wizardkette hat der Endanwender jederzeit die Möglichkeit, Korrekturen vorzunehmen. Am Ende der Wizardkette kommt es zum bereits be-

schriebenen lokalen Schreibvorgang in die Datenbank der Verbundanwendung, ehe in einem nachgelagerten automatisierten Prozessschritt die Verbuchung in die Backend-Systeme erfolgt.

Als Alternative zum Wizard bietet sich die Verwendung von Laschen oder auch Reitern an. Dabei wird eine komplexe Benutzeroberfläche in mehrere Laschen unterteilt, wobei jede einzelne Lasche, wie auch schon im Wizard-UI, zusammengehörige Daten bündelt. Der Unterschied zum Wizard-UI liegt bei der Laschen-Technik bei dem wahlfreien Zugriff auf jede einzelne Maske. Musste beim Wizard durch Vorwärts- bzw. Rückwärtsnavigation die jeweilige Maske angesteuert werden, kann der Endanwender jederzeit wahlfrei einen beliebigen Reiter auswählen und die notwendigen Eingaben vornehmen. Diese Technik bietet sich immer dann an, wenn die Daten der Masken unabhängig voneinander sind. Im Gegensatz dazu ist der Einsatz der Wizard-Technik immer dann von Vorteil, wenn der Inhalt einer nachgelagerten Maske von vorangegangenen Eingaben abhängig ist.

Welche Technik auch immer bevorzugt wird, sie helfen bei der Strukturierung komplexer, rollenorientierter Eingabeoberflächen, wie sie in Verbundanwendungen häufig anzutreffen sind, und verhindern dadurch die Verwendung zweier aufeinanderfolgender Benutzeraufgaben für ein und dieselbe Rolle innerhalb eines Prozesses.

Wie bereits mehrfach erwähnt wurde, sind aus Benutzeroberflächen heraus lediglich synchrone Aufrufe durchzuführen, damit die Akzeptanz der Verbundanwendung mit kurzen Antwortzeiten steigt. Aber auch für synchrone Aufrufe gilt selbstverständlich die Definition eines Servicevertrags. Dessen Implementierung ist allerdings auf Performanz ausgelegt. Damit empfiehlt sich auch die Verwendung der performantesten Technologie, die zur Anbindung der involvierten Systeme geeignet ist. Stellt sich während der Lasttests jedoch kein befriedigendes Antwortzeitverhalten ein, muss über die Verwendung von Caching als mögliche Optimierung entschieden werden. Reicht auch diese Maßnahme nicht aus, bleibt als Alternative die lokale Kopie der Daten in der Datenbank der Verbundanwendung. Wie konkrete Implementierungen von Verbundanwendungen in der Praxis zeigen, ist diese Vorgehensweise durchaus keine Seltenheit. Insbesondere Stammdaten (z.B. Kunden-, Lieferanten-, Produktstammdaten) bieten sich hierfür an, da sie sich in der Regel selten ändern und sich insbesondere in großen Unternehmen ohnehin Prozesse zur Verteilung von Stammdatenänderungen

etabliert haben, so dass das Hinzufügen eines neuen Abnehmers von Änderungen keinen zu großen Aufwand bedeutet.

Die Anforderung zur Verwendung rein synchroner Aufrufen aus den Benutzeroberflächen heraus kann zu Problemen führen, wenn die benötigte Funktionalitäten seitens der Backend-Systeme lediglich asynchron zur Verfügung gestellt werden. In diesen Fällen hat sich die Verwendung einer Synchron-Asynchron-Brücke bewährt, wie sie in Abbildung 68 in Form eines BPMN-Patterns dargestellt ist.

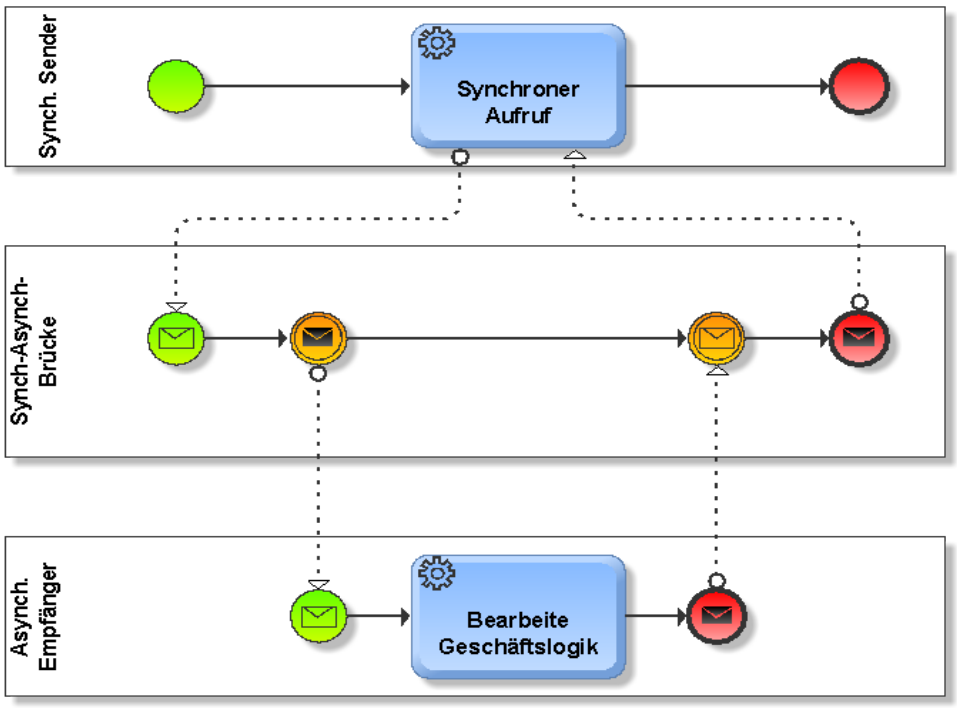



***Abbildung 68: Synchron-Asynchron-Brücke***

Der synchrone Sender fordert einen Dienst an, der lediglich asynchron vom Backend geleistet werden kann. In Abbildung 68 ist zu erkennen, wie die Synchron-Asynchron-Brücke technisch die Verbindung zum synchronen Aufrufer hält, während sie selbst asynchron den Empfänger aufruft, um anschließend am nachrichtenbasierten Zwischenereignis in bewährter Manier auf die Antwort des Backend-Systems wartet. In SAP NetWeaver BPM wird dieses Verhalten beispielsweise dadurch erreicht, indem den nachrichtenbasierten Start- und Endereignissen der Synchron-Asynchron-Brücke *dieselbe* synchrone Schnittstelle zugeordnet werden. Dadurch wird dem Framework signalisiert, dass der Prozess synchron aufgerufen wird und dem entsprechend die Verbindung zum Aufrufer nach Übergabe der Daten nicht abgebaut werden darf.

Unmittelbar nach Empfang der Antwort vom asynchronen Empfänger wird das Ergebnis schließlich an den wartenden Aufrufer weitergeleitet.

Der umgekehrte Fall ist natürlich auch denkbar: seitens der Verbundanwendung wird eine asynchrone Verarbeitung vorausgesetzt, das involvierte Backend-System liefert seine Dienste jedoch nur synchron ab. In diesem Szenario bietet sich die Verwendung der Asynchron-Synchron-Brücke an, die wiederum als BPMN-Pattern in Abbildung 69 dargestellt ist.

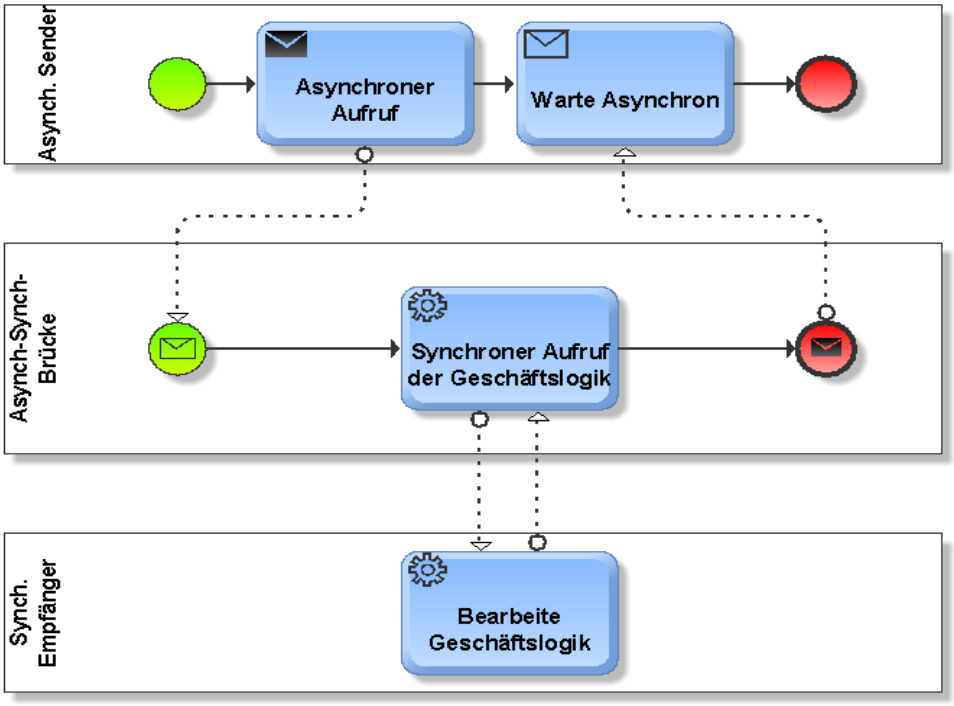



*Abbildung 69: Asynchron-Synchron-Brücke*

Diesmal setzt der Sender einen asynchronen Aufruf ab und begibt sich an der empfangenden Aufgabe in Wartestellung. Die Asynchron-Synchron-Brücke wiederum ruft die Geschäftslogik synchron im Backend-System ab, packt die erhaltene Antwort in einer Nachricht zusammen, die wiederum den Sender aus seinen Wartezustand befreit.

Offensichtlich vermitteln beide Brücken zwischen der Verbundanwendung einerseits und den Backend-Systemen andererseits. Damit gehören sie zum Implementierungsrepertoire der Servicevertrag-Implementierungsschicht. Auf diese Weise können die unterschiedlichen synchronen bzw. asynchronen Kommunikationsanforderungen der Composite Application umgesetzt werden.

Zum Abschluss dieses Abschnitts sei noch auf ein weiteres Verfahren verwiesen, wie die Herausforderungen der losen Kopplung mit Benutzeroberflächen in Einklang gebracht werden können. Sowohl in Erl 2008 als auch in Maier et. al. 2009c wird auf das UI-Mediator-Pattern hingewiesen, dass die Verwendung eines sogenannten Service-Human-Interaction-Layers (SHIL) empfiehlt. Dieser Layer simuliert gegenüber dem wartenden Endbenutzer Synchronität, indem es ihm während der Ausführung der asynchronen Backend-Aufrufe Masken vorblendet, die ihm den Status der Backend-Abfrage anzeigt. Wann diese Masken angezeigt werden, ist von dem Antwortzeitverhalten der beteiligten Backend-Systeme abhängig. Sollte die Antwort innerhalb kürzester Zeit (z.B. innerhalb von 2 Sekunden) eintreffen, wird diese ohne Zwischenschaltung eines Wartedialogs unmittelbar zur Anzeige gebracht. Verstreichen allerdings die 2 Sekunden ohne Antwort, blendet der SHIL einen Dialog auf, der den Anwender über die Verzögerung informiert. Verstreicht auch ein zweiter Zeitpunkt, z.B. nach 12 Sekunden, so tritt ein Timeout ein und der Anwender erhält in einem zweiten Dialog die Information, dass der Dienst zur Zeit nicht zur Verfügung steht.

Im Wesentlichen verbirgt sich hinter dem SHIL eine Synchron-Asynchron-Brücke mit erweiterter Funktionalität zur Steuerung des Dialogs mit dem Endanwender. In einer komfortableren Variante arbeitet der SHIL auch über Prozessschritte hinweg, wenn also tatsächlich zwei aufeinanderfolgende Schritte ein und derselben Rolle als zwei separate Benutzeraufgaben modelliert wurden. Normalerweise müsste der Endanwender über seine Arbeitsliste zum zweiten Schritt navigieren. Wird der SHIL hingegen befähigt, auch mit der Arbeitsliste, beispielsweise über ein API, zusammenzuarbeiten, kann der nächste interaktive Schritt für denselben Benutzer unmittelbar zur Anzeige gebracht werden, ohne dass dieser nochmals seine Arbeitsliste bemühen müsste. Dadurch wird der Komfort für den Anwender natürlich erhöht. Es verleitet allerdings auch zur Modellierung wenig effizienter Prozesse, zu deren Umsetzung eigentlich die Wizard- sowie die Laschen-Technik besser geeignet sind.

## 5.6 Pattern

Bei der Gestaltung neuer Applikationen sind Pattern in der heutigen Informatik und insbesondere im Software-Engineering nicht mehr wegzudenken. Sie repräsentieren in der Praxis bewährte Lösungen für einen bestimmten Problembereich. Wegweisend

ist in dieser Hinsicht sicherlich das Werk der „Gang-of-Four“ (GoF) über Design Pattern für die objektorientierte Softwareentwicklung (Gamma et. al. 1995).

Verbundanwendungen können in vielerlei Hinsicht von Pattern profitieren, da sie aufgrund ihres integrativen Charakters viele Berührungspunkte zu unterschiedlichsten Problembereichen besitzen. Auch im Rahmen dieser Arbeit wurden bereits patternähnliche Lösungen behandelt, wie beispielsweise die Diskussionen im vorangegangenen Abschnitt über die Synchron-Asynchron-Brücke, über die unterschiedlichsten UI-Techniken oder über die verschiedenen Prozesslösungen mittels BPMN in Kapitel 4.1 gezeigt haben. Es sollen an dieser Stelle jetzt nicht die einzelnen in der Literatur besprochenen Pattern auf ihre Relevanz für die verschiedenen Schichten einer Verbundanwendung hin untersucht werden. Es wird vielmehr Wert auf die Behandlung von Pattern für die Servicevertrag-Implementierungsschicht gelegt, da sie bei der Entwicklung von Verbundanwendungen eine zentrale Funktion übernimmt. In ihr werden technische Prozesse abgewickelt, weshalb im weiteren Verlauf dieses Kapitels verschiedene Prozessfragmente für unterschiedlichste Problemstellungen behandelt werden. Als Modellierungssprache zur Umsetzung der Pattern wird einmal mehr die BPMN verwendet.

Aus der Beschreibung der Architektur von Verbundanwendungen wurde deutlich, dass die Servicevertrag-Implementierungsschicht von der Composite über die Service-Schnittstelle einen konkreten Auftrag bekommt, nämlich eine bestimmte betriebswirtschaftliche Funktionalität zu erbringen. Diese wird unter Wiederverwendung existierender Funktionalitäten der Altsysteme erbracht, indem die Implementierungsschicht aktiv Informationen aus den Backend-Systemen zusammenträgt bzw. dort Aktivitäten oder Standardprozesse anstößt. Die Implementierungsschicht übernimmt folglich einen aktiven Part. Derartige Pattern werden im Abschnitt *Composition Pattern* behandelt.

Im Gegensatz zu den Composition Pattern stehen die eher passiven s*ystemzentrischen Pattern*. Sie werden durch bestimmte Nachrichten seitens einer Composite instanziiert, gehen aber anschließend nicht aktiv auf die Backend-Systeme zu, um Informationen abzugreifen, sondern warten passiv auf entsprechende Nachrichten, die aufgrund festgelegter betriebswirtschaftlicher Abläufe zwangsläufig auftreten müssen, entnehmen ihnen die für die Verbundanwendung wichtigen Informationen und schicken sie schließlich an die Composite.

Wie bei Pattern üblich, wird im Folgenden das mit dem Pattern adressierte Problem kurz skizziert und anschließend die mögliche Lösung anhand eines Prozessfragments erläutert.

In diesem Abschnitt werden keine Workflow-Pattern behandelt, wie sie typischerweise für die benutzerzentrischen Prozesse der Verbundanwendung benötigt werden (Business Composition). Sie sind sowohl theoretisch in Form von Workflow-Pattern durch van der Aalst et. al. 2003, als auch in deren Umsetzung in BPMN-Diagramme bzw. in UML 2.0 Aktivitätsdiagramme durch White 2004 ausführlichst diskutiert worden. Eine Umsetzung der Workflow-Pattern in ausführbare Prozesse für die Ablaufumgebung SAP NetWeaver BPM, die für die Umsetzung des Proof-of-Concepts verwendet wurde, findet sich in Balko 2009 bzw. Balko 2010b. Weitere ergiebige Quellen zu Pattern insbesondere im Zusammenhang mit SOA, Unternehmensarchitekturen und Integrationslösungen finden sich in Erl 2008, Fowler 2002 bzw. Hohpe & Woolf 2004 respektive. Eine aufschlussreiche Diskussion über Pattern speziell in der Wirtschaftsinformatik mit weiteren Literaturreferenzen findet sich schließlich in Winter 2009.

### *5.6.1 Composition Pattern*

Dieser Abschnitt behandelt die verschiedenen Composition Pattern, die innerhalb der Servicevertrag-Implementierungsschicht Verwendung finden. Sie zeichnen sich durch eine aktive Rolle der Implementierungsschicht aus, d.h. nach Übergabe der Daten wird aktiv die Erfüllung des Vertrags durch Aufruf von Servicedienstleistern erbracht.

Das erste Pattern adressiert den einfachsten Fall der Service-Erfüllung durch Weiterleitung an genau ein System, das die erforderte Dienstleistung vollständig erbringen kann. Es handelt sich dabei um das klassische Request-Reply-Integrationspattern (Hohpe & Woolf 2004, S. 154). Es ist in Abbildung 70 dargestellt.

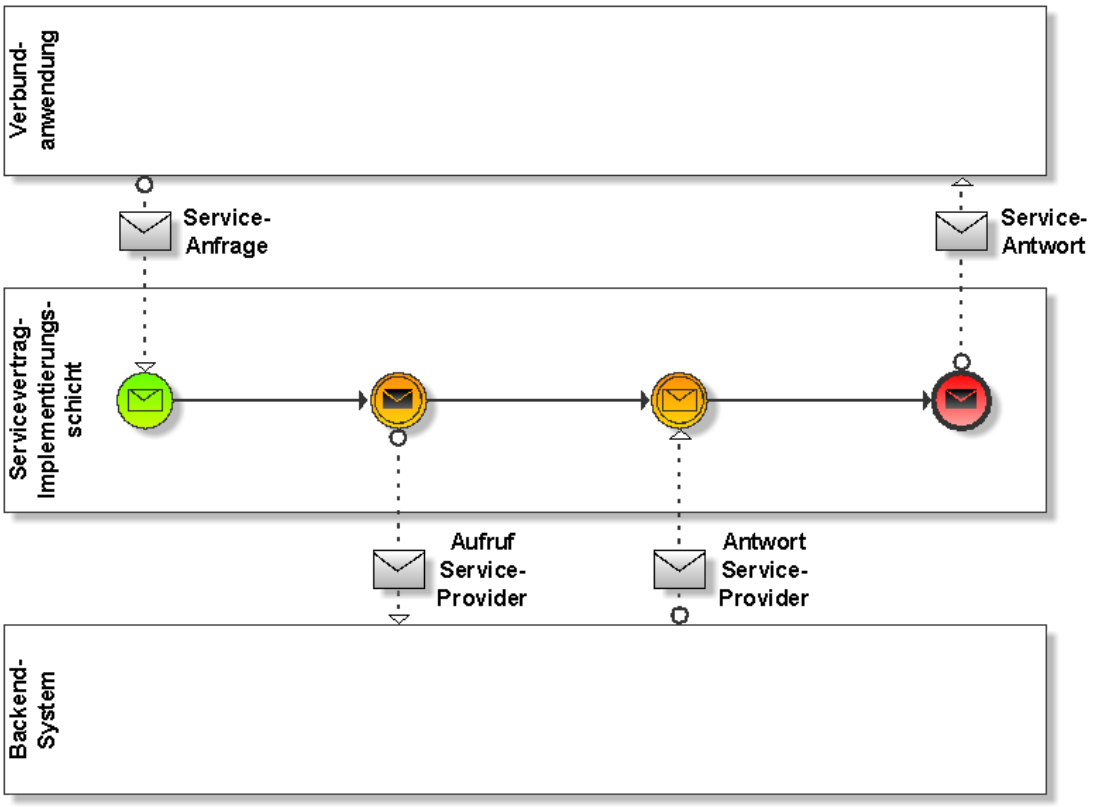



*Abbildung 70: Request-Reply-Integrationspattern*

Die Verbundanwendung sendet eine Nachricht gemäß des vereinbarten Vertrags an die Implementierungsschicht. Diese wiederum leitet sie asynchron an das betroffene Backend-System weiter und wartet anschließend auf dessen Antwort. Auch hier kommt wieder das nachrichtenbasierte Zwischenereignis zum Einsatz. Nach Erhalt der Antwort wird diese an die wartende Composite durchgereicht. Neben der eigentlichen Nachrichtenvermittlung übernimmt diese Schicht auch Protokoll- und Datenformatsanpassungen. So wird die von der Composite eintreffende Nachricht aus dem kanonischen Datenmodell der Verbundanwendung in das Zielformat der Zielapplikation gewandelt. Dabei finden sowohl Struktur- als auch Datenmappings statt. Strukturmappings vermitteln zwischen unterschiedlichen Datenstrukturen, wobei semantisch gleiche Felder zugeordnet werden. Dabei können die Felder zwar in den differierenden Datenstrukturen direkt zugeordnet werden – sie sind aber in den Quell- und Zielstrukturen entweder unterschiedlich benannt oder sie befinden sich auf unterschiedlichen hierarchischen Ebenen. Datenmappings werden immer dann benötigt, wenn Datentransformationen gemäß bestimmter Konvertierungsvorschriften durchgeführt werden müssen. Die einfachste Form der Transformation ist der Wechsel zwischen verschie-

denen Datentypen wie von String zu Float oder von Date nach String. Ein weiteres Beispiel ist das Zusammenfügen oder Trennen von Feldern: so kann in der eintreffenden Nachricht für eine Kundenstruktur der Vor- und Nachname getrennt sein, während das Zielformat die beiden Namen in einem Feld bereitgestellt haben möchte. Schließlich ist es auch denkbar, dass die Feldinhalte komplett geändert werden müssen. So kann die Anrede in einem Quellsystem lediglich durch fortlaufende Nummern dargestellt werden (1=Frau, 2=Herr), während das Zielsystem eine Klartextdarstellung verlangt. Diese und ähnliche Aufgaben sind ebenfalls in der Servicevertrag-Implementierungsschicht zu realisieren.

In obigen Beispiel wird lediglich ein System zur Erbringung des Dienstes benötigt. Sind hingegen mehrere Systeme beteiligt, so ändert sich der Ablauf wie in Abbildung 71 dargestellt.

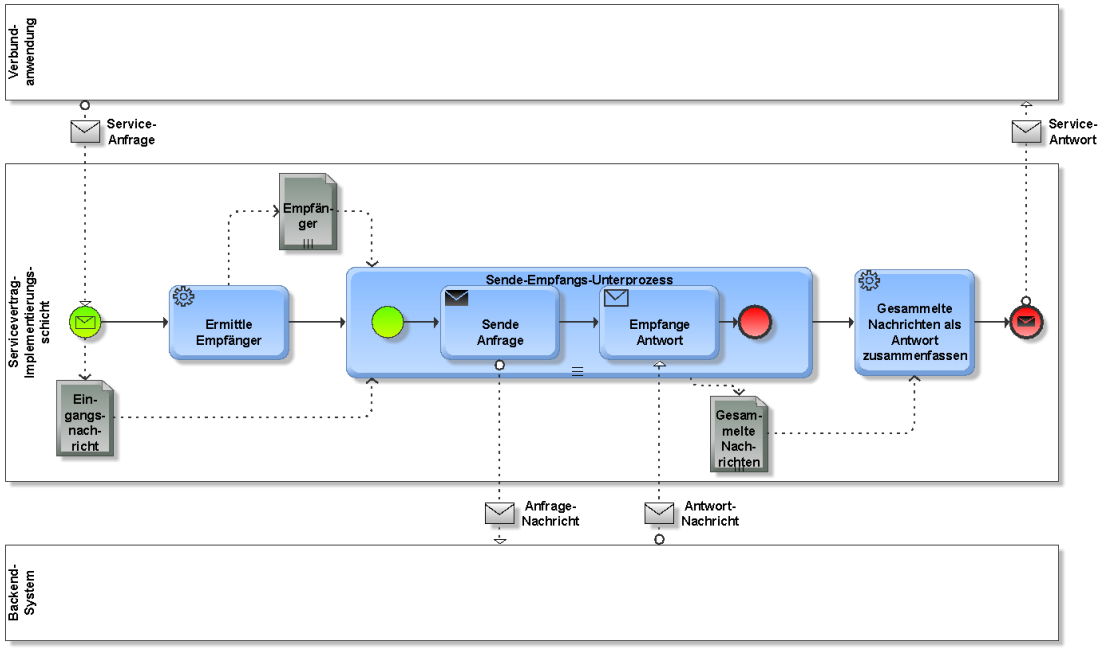



***Abbildung 71: Request-Reply-Integrationspattern unter sequenzieller Einbeziehung mehrerer Systeme***

Zu einer Eingangsnachricht müssen mehrere Systeme getriggert werden, um den erwünschten Service zu erbringen. Die Liste der Empfänger wird dabei zur Laufzeit ermittelt und in einem Datenobjekt gespeichert. Anschließend werden die Systeme nacheinander abgefragt, wie dies durch die sequenzielle Mehrfachausführung des Unterprozesses verdeutlicht wird. Die beteiligten Backend-Systeme werden dabei mit ein und demselben Datenformat angesprochen und sie antworten auch wieder in einem

einheitlichen Format. Dargestellt ist dies in dem Modell durch die Nachrichten *Anfrage-Nachricht* bzw. *Antwort-Nachricht*. Sinnvoll ist dieses Modell, wenn es sich bei den angesprochenen Backends um gleichartige Systeme handelt, beispielsweise um SAP-Systeme mit demselben Release-Stand, die sich jedoch in unterschiedlichen Regionen befinden. Nach dem Empfang aller Nachrichten werden diese zu einer Antwortnachricht für die Verbundanwendung verpackt und versendet.

In diesem Szenario wird von einer sequenziellen Verarbeitung ausgegangen. Ist hingegen die Verarbeitung in den Backends vollständig unabhängig, können diese auch parallel angesprochen werden. In BPMN wird dies durch eine parallele Mehrfachausführung dargestellt, wie dies Abbildung 72 veranschaulicht.

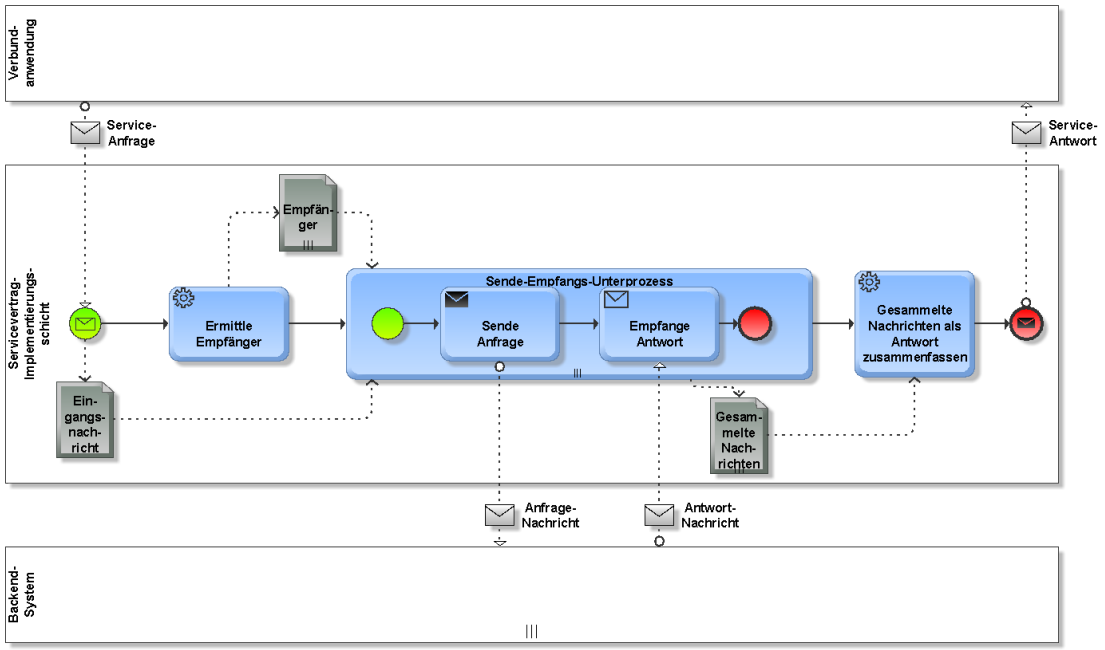



***Abbildung 72: Request-Reply-Integrationspattern unter paralleler Einbeziehung mehrerer Systeme***

Zu erkennen ist zudem die parallele Ausführung in den Backend-Systemen, die jetzt ebenfalls mit der Mehrfachinstanziierung gekennzeichnet sind. Nach wie vor wird auch in diesem Pattern von gleichartigen Nachrichten für die angesprochenen Systeme ausgegangen. In typischen Szenarien für Verbundanwendungen wird dies so allerdings nicht umzusetzen sein, da in den Unternehmen aufgrund ihrer Historie eine höchst heterogene Landschaft und damit unterschiedliche Schnittstellen und Protokolle angetroffen werden. Zur Lösung dieses Problems eignet sich das in Abbildung 73 dargestellte Pattern.

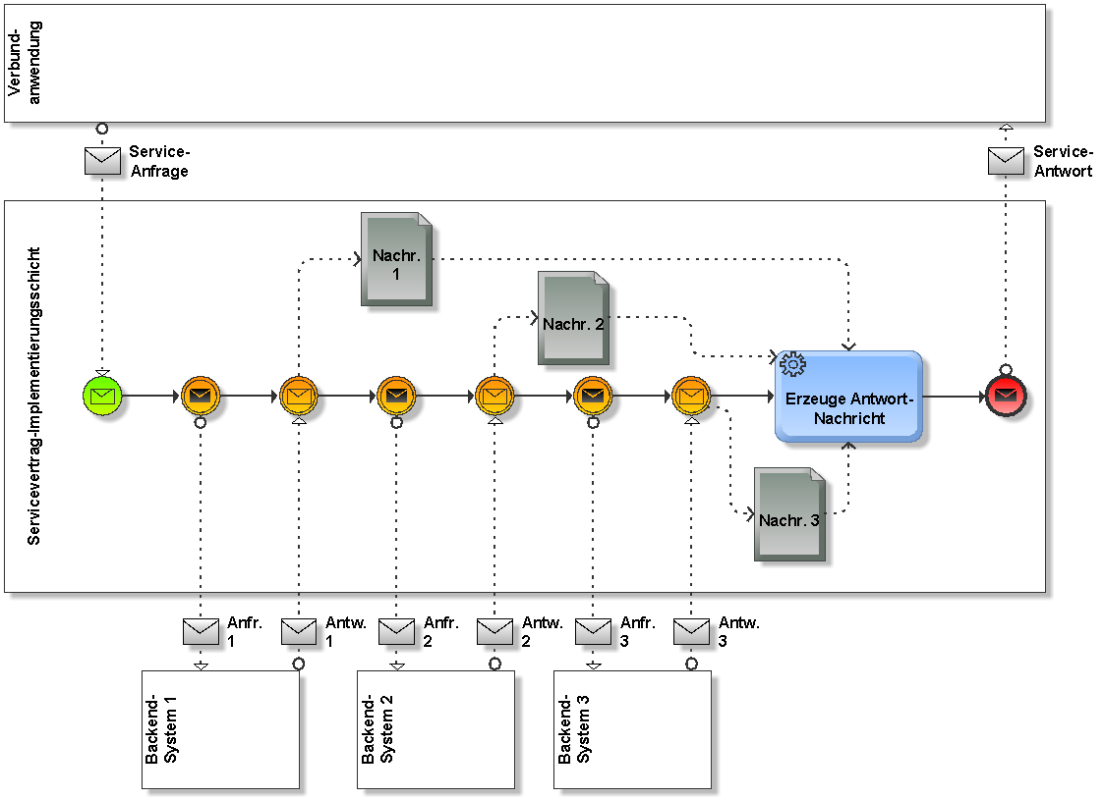



***Abbildung 73: Request-Reply-Integrationspattern unter sequenzieller Einbeziehung mehrerer Systeme mit unterschiedlichen Schnittstellen***

Die unterschiedlichen Backend-Systeme werden jetzt über unterschiedliche Schnittstellen angesprochen. Das Pattern zeigt wiederum zunächst die sequenzielle Verarbeitung. Sie ist immer dann anzuwenden, wenn zwischen den Aufrufen Abhängigkeiten existieren. So verwendet beispielsweise der Aufruf von Backend-System 2 ein Feld der Antwort von Backend-System 1. Ein Aufruf ohne diese Information wäre zwecklos und ist daher zwingend nach dem Aufruf von Backend 1 durchzuführen. Sind derartige Abhängigkeiten nicht gegeben, so ist die parallele Verarbeitung aufgrund von Performance-Vorteilen stets zu bevorzugen. Abbildung 74 zeigt das Pattern in BPMN-Darstellung.

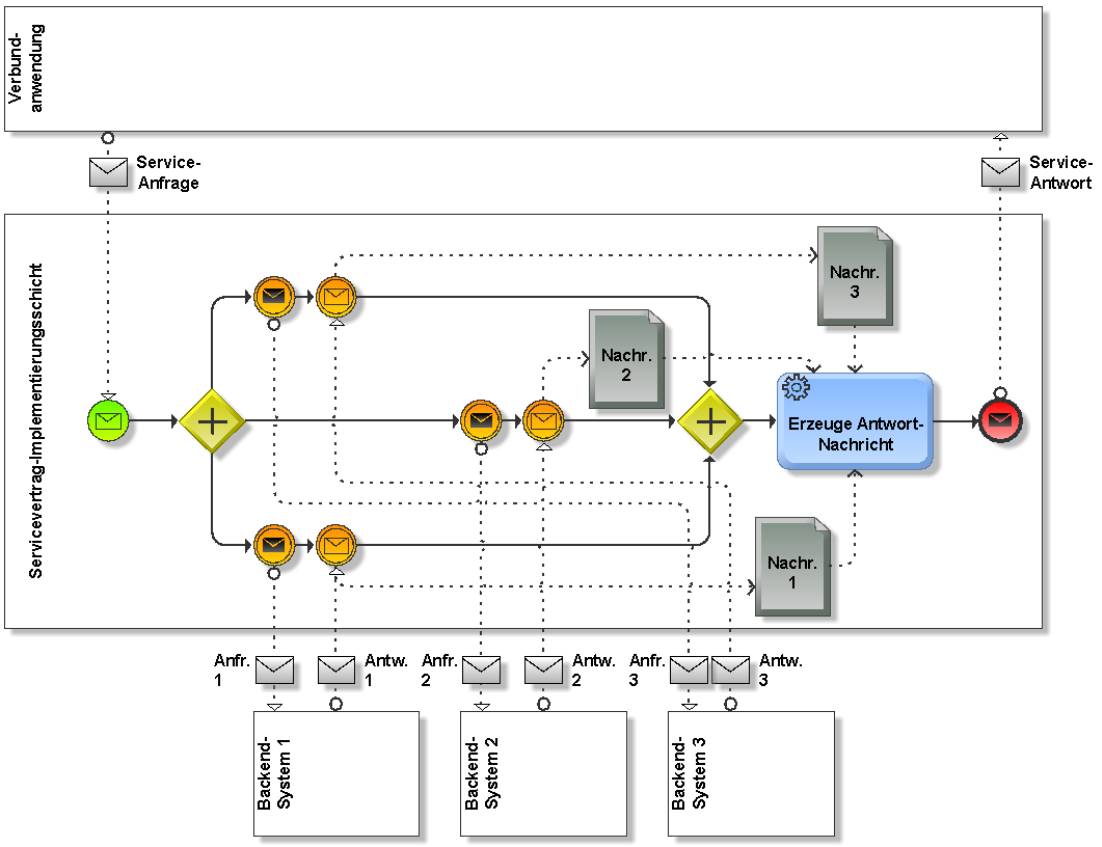



***Abbildung 74: Request-Reply-Integrationspattern unter paralleler Einbeziehung mehrerer Systeme mit unterschiedlichen Schnittstellen***

Durch das parallele Gateway wird der Sequenzfluss in allen drei ausgehenden Pfaden gleichzeitig gestartet und die Backend-Systeme über unterschiedliche Schnittstellen angesprochen, wie dies durch die drei unterschiedlichen Anfragen und Antworten im Modell ausgedrückt wird. Das parallele Gateway am Ende der drei Sende-Empfangs-Pfade erzwingt die Fortsetzung des Sequenzflusses mit der Erzeugung der Antwortnachricht erst nach Eintreffen sämtlicher Antworten der beteiligten Backend-Systeme. Dieses Pattern ist dem sequenziellen Pattern aufgrund der Performanzvorteile vorzuziehen, sofern es das umzusetzende Szenario zulässt. Letztendlich entscheidet allein das Abhängigkeitskriterium über die jeweilige Verwendung.

Ein weiterer Vorteil dieses Pattern ist die implizite Lösung des Problems, dass die Antwortnachrichten nicht unbedingt in der Reihenfolge eintreffen, wie sie ursprünglich gesendet wurden. In der klassischen Literatur zu Enterprise Integration Patterns findet sich dazu das *Resequencer*-Pattern (Hohpe & Woolf 2004, S. 283), das sich explizit dieser Problematik annimmt. Durch Verwendung des parallelen Gateways oder, wie noch an einem weiteren Beispiel zu sehen sein wird, des inklusiven Gateways löst sich das Problem automatisch: da bei den genannten Gateways parallel auf unter-

schiedlichste Nachrichten gewartet wird und die Pfade an dem zusammenführenden Gateway synchronisiert werden, ist in dem Sequenzfluss, der dem Gateway folgt, der Empfang sämtlicher erforderlicher Nachrichten sichergestellt, egal in welcher Reihenfolge sie nun eingetroffen sind. Im Gegensatz zum *Resequencer*, bei dem die eingegangenen Nachrichten anschließend einzeln in anderer Reihenfolge verschickt werden, wird bei dem hier präsentierten Composite Pattern zwar nur eine Nachricht erzeugt und verschickt, die Problematik der unterschiedlich eintreffenden Nachrichten ist aber bei beiden Pattern dieselbe.

Ist die Unabhängigkeit der Aufrufe gegeben und will der Modellierer zusätzliche Flexibilität in seine Lösung berücksichtigen, da beispielsweise abhängig vom Inhalt der Eingangsnachricht unterschiedliche Systeme parallel angesprochen werden sollen, so bietet sich der Einsatz von Regelwerken an, wie die in Abbildung 75 dargestellte Variation des parallelen Szenarios veranschaulicht.

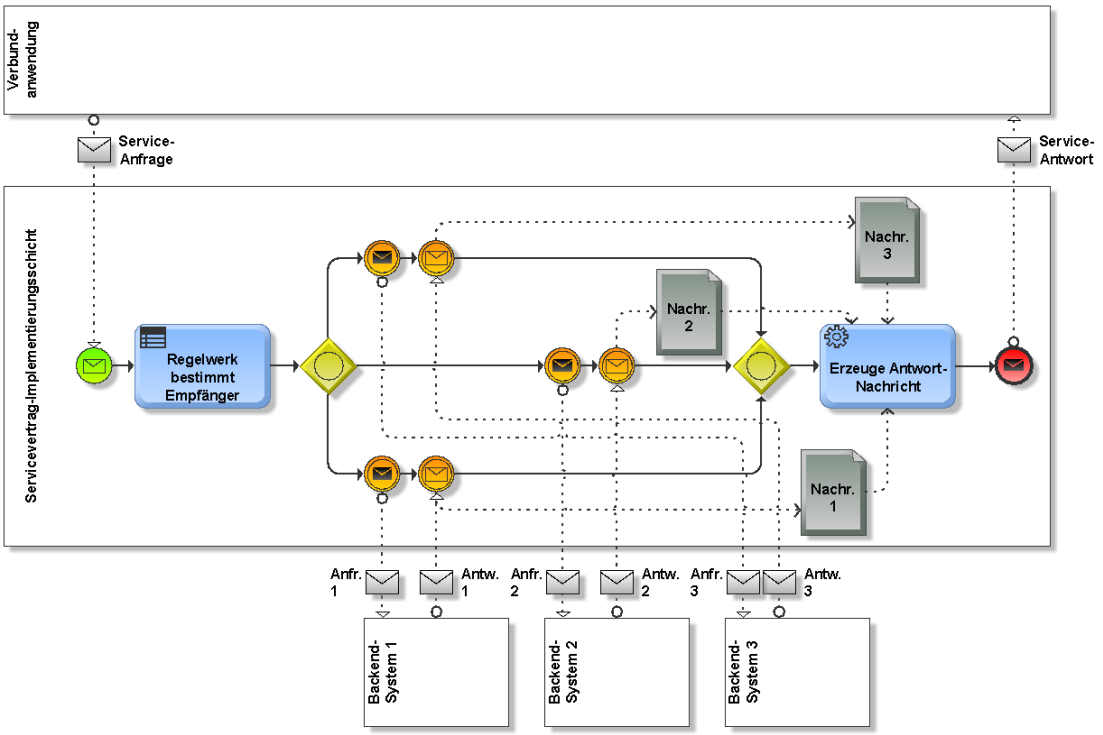



***Abbildung 75: Einsatz eines Regelwerks zur Bestimmung der Empfänger im Request-Reply-Pattern***

Die Aufgabe vom Typ *Geschäftsregel* bestimmt aufgrund des Nachrichteninhalts den weiteren Verlauf, indem es für das darauffolgende inklusive Gateway die zu involvierenden Systeme ermittelt. Das Gateway reicht folglich nur Token an die Pfade weiter, deren dazugehörige Bedingung mit der von der Geschäftsregel ermittelten übereinstimmt. Dabei muss die Bedingung für einen Pfad nicht kompliziert sein. Kon-

kret könnte die Regel eine Liste von Systemnummern erzeugen. Jede Systemnummer entspricht exakt einem Pfad, den das inklusive Gateway verlässt. Folglich lautet die Bedingung für jeden Pfad lediglich, ob die dem Pfad repräsentierende Systemnummer in der Liste vorhanden ist.

Die Zusammenführung der parallel aktivierten Pfade erfolgt wiederum durch ein inklusives Gateway. Die BPMN-Spezifikation schreibt bei der Semantik des zusammenführenden inklusiven Gateways vor, dass erst dann ein Token an den ausgehenden Pfad weitergereicht wird, wenn alle ursprünglich gestarteten Pfade abgeschlossen wurden. Genau dieses Verhalten ist an dieser Stelle erwünscht.

Durch den Einsatz eines Regelwerks wird zusätzliche Flexibilität für ausführbare Prozesse gewonnen. Da mit diesem Ansatz weitere Vorteile für Verbundanwendungen verbunden sind, wird diesem Aspekt in einem eigenständigen Abschnitt 5.7 besondere Aufmerksamkeit gewidmet.

In den bisherigen Composition Pattern antworteten die asynchron kontaktierten Systeme unmittelbar auf die Anfragen der Servicevertrag-Implementierungsschicht mit einer passenden, ebenfalls asynchron verschickten, Antwortnachricht. Doch wie ist zu verfahren, wenn Systeme zwar Schreib- bzw. Änderungsoperationen per Schnittstelle anbieten, jedoch nicht selbst *unmittelbar* und *aktiv* mit Nachrichten darauf reagieren? Wie kann in diesen Fällen die Verbundanwendung aus ihren Wartezustand befreit werden? Die Lösung liegt in dem zyklischen Abfragen von Zuständen der geänderten Geschäftsobjekte in den betroffenen Backend-Systemen. So bietet jedes Geschäftssystem in der Regel Such- und Leseoperationen auf die von ihm verwalteten Objekte an, meistens sogar synchron. Die Implementierungsschicht wird daher über diese Leseschnittstelle den Inhalt des betroffenen Objektes nachlesen und überprüfen, ob die letzten Änderungen bereits verarbeitet wurden. Vereinfacht wird diese Überprüfung durch Statusfelder, die üblicherweise Bestandteil des Objekts sind. Ist der gewünschte Zustand schließlich eingetroffen, kann die Verbundanwendung über den Erfolg der Operation informiert werden. Da dieses Warten auf das Eintreten des gewünschten Zustands durchaus mit mehreren Leseaufrufen verbunden sein kann, handelt es sich bei diesem Pattern um ein klassisches Polling (vgl. auch Abschnitt 5.3 über den Einsatz von Ereignissen, um die Verbundanwendung aus dem Wartezustand zu befreien. Auch dort wurde Polling als probates Mittel bereits eingesetzt).

Konkret lässt sich dieses Verhalten beim Anlegen bzw. Ändern eines Auftrags verfolgen: das Backend-System wird asynchron über den neuen Auftrag informiert, es erzeugt allerdings selbst keine eigene Quittierungsnachricht. Folglich wird das Backend in regelmäßigen Abständen gepollt – bei Neuanlagen über die Suchschnittstelle, da aufgrund der asynchronen Verarbeitung noch keine Auftragsnummer existiert, die referenziert werden könnte, und bei Änderungen über die Leseschnittstelle mit der Auftragsnummer als Übergabeparameter. Bei der Suchschnittstelle muss durch geeignete Suchparameterkombination die Eindeutigkeit des Suchtreffers auf den neu angelegten Auftrag gewährleistet sein. Bei Bestellungen ist dies oftmals schon allein durch eine eindeutige Besteller-Referenznummer gewährleistet, die der Bestellung mitgegeben wird. Liefert die Suche selbst das gewünschte Objekt nicht direkt zurück, müssen folglich die Daten des Auftrags auch in diesem Fall nochmals nachgelesen werden. Anschließend sind vielfältige Kriterien denkbar, anhand derer der Erfolg- oder Misserfolg der ursprünglich veranlassten Operation festgestellt werden kann. Dies ist natürlich stark von dem beteiligten System abhängig, doch lassen sich eine oder mehrere der folgenden Kriterien heranziehen:

- Überprüfung eines Status-Feldes: hat der Auftrag mittlerweile den erwarteten Zustand eingenommen?
- Überprüfung des Fehler-Status: hat sich aufgrund der letzten Änderung ein Fehler ergeben? Wenn ja, so ist die dazugehörige Fehlermeldung zu analysieren.
- Überprüfung der Auftragshistorie: enthält die Auftragshistorie die letzte Auftragsänderung?
- Überprüfung konkreter Felder: enthält der Auftrag die eingebrachten Änderungen?

Bei der Umsetzung des Pollings in BPMN stehen nun wieder mehrere Varianten zur Auswahl. Abbildung 76 zeigt nun eine erste Variante, bei der das betroffene System synchron angefragt wird und die Leseschleife explizit ausmodelliert ist.

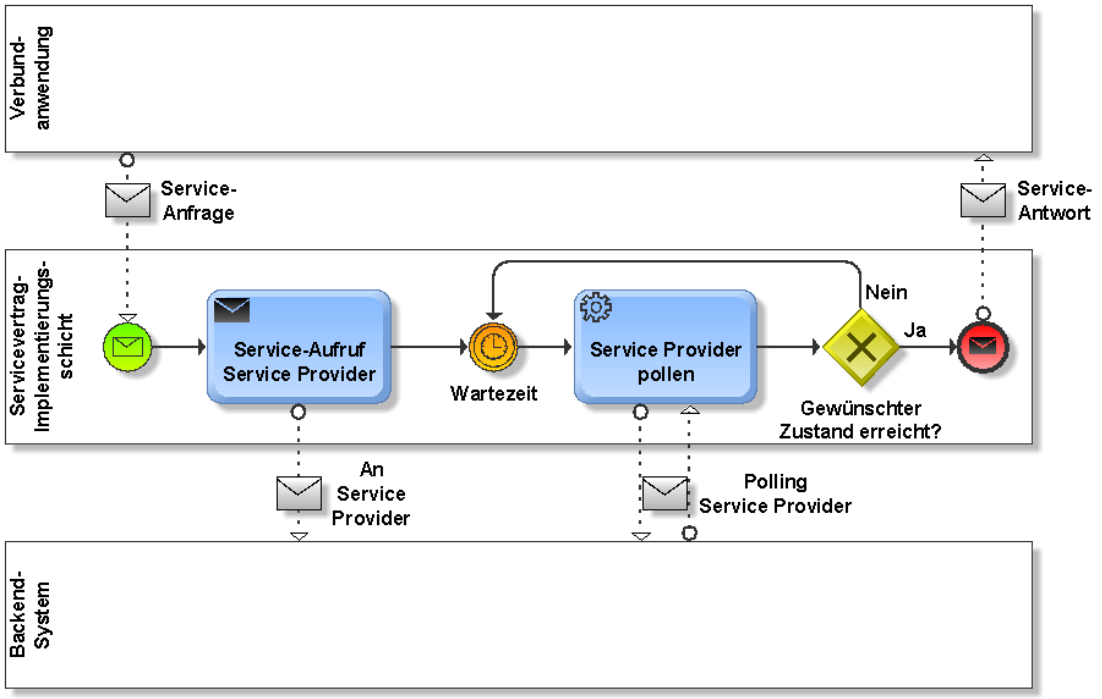



*Abbildung 76: Synchrones Polling mit expliziter Leseschleife*

Getriggert wird der Ablauf durch die eingehende Nachricht der Verbundanwendung. Die Service-Anfrage wird wie gewohnt an die Geschäftsanwendung asynchron durchgereicht. Anschließend begibt sich der Prozess in einen Wartezustand, um dem Backend Zeit zur Verbuchung der Anfrage zu geben. An dieser Stelle könnte natürlich auch unmittelbar das Polling erfolgen. Allerdings ist die Wahrscheinlichkeit einer negativen Antwort aufgrund erhöhter Laufzeiten im Backend recht groß, da Services per Definition eher grobgranular geschnitten sind und einen gewissen betriebswirtschaftlichen Umfang abdecken. So ist es bei einer Bestellung eben nicht damit getan, einen Eintrag in einer Datenbanktabelle zu erzeugen. Vielmehr wird beispielsweise eine Bestandsüberprüfung angestoßen, gegebenenfalls die Produktion veranlasst, Lieferanten involviert usw. Letztendlich muss die Entscheidung über die Positionierung des Timers im BPMN-Modell von dem jeweiligen Szenario und der konkreten Funktionalität des zuvor gerufenen Services abhängen.

Nach Ablauf der Wartezeit erfolgt der synchrone Aufruf zur Feststellung des Objekt-Zustands. Im Anschluss stellt das exklusive Gateway das Eintreten des gewünschten Zielzustands fest. Ist er erreicht, kann die Verbundanwendung aus ihrer Wartesituation befreit werden. Andernfalls wird erneut am Timer-Zwischenereignis gewartet und der Polling-Zyklus beginnt von vorn.

Eleganter und mit mehr Möglichkeiten zur Fehlerüberwachung der eigentlichen Polling-Schleife erweist sich die Modellierung mittels Teilprozessen, wie dies in Abbildung 77 dargestellt ist.

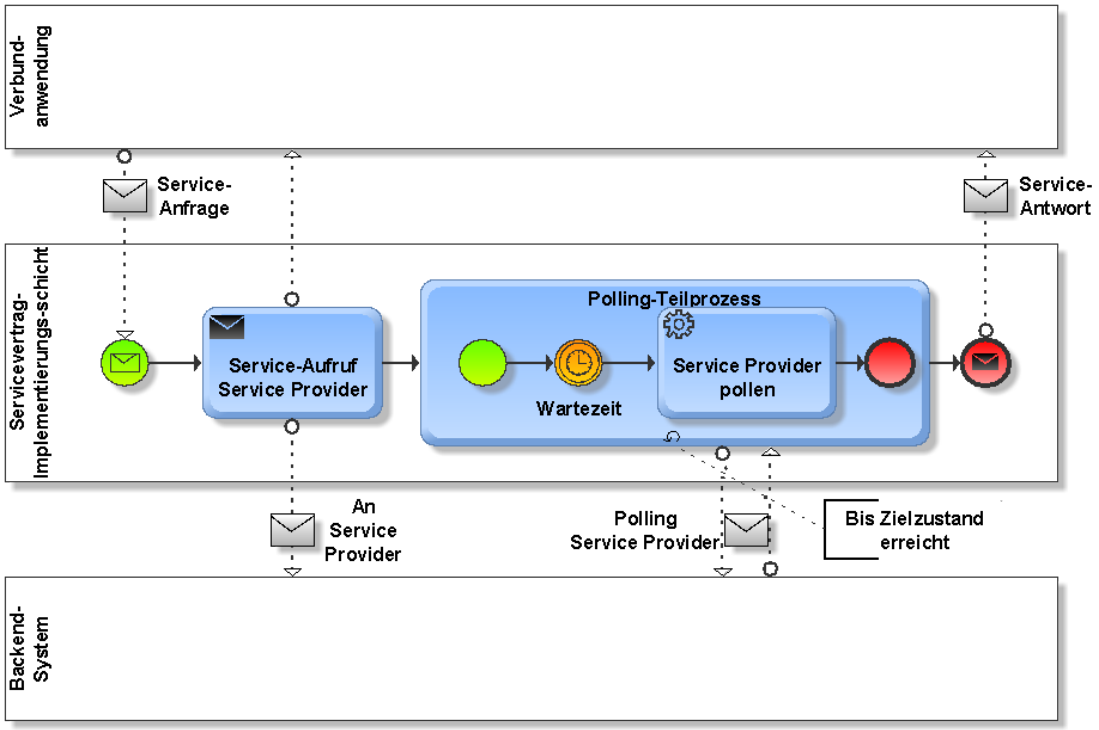



*Abbildung 77: Synchrones Polling unter Verwendung eines Teilprozesses mit sequenzieller Schleife*

Kennzeichnend ist die Einbettung des Polling-Vorgangs in einen eigenständigen Teilprozess. Er ist mit einem sequenziellen Schleifensymbol versehen, da die Anzahl der Schleifendurchläufe vorher unbekannt ist und erst durch Erfüllung des Zielzustands bestimmt wird. Die Verwendung des Teilprozesses ermöglicht weitere Verbesserungen dahingehend, als dass durch angeheftete Ereignisse eine bessere Überwachung des Pollings ermöglicht wird und Fehlerfälle besser behandelt werden können. Passende Fehlerbehandlungsstrategien wurden im Rahmen dieser Arbeit bereits mehrfach diskutiert, wie beispielsweise in Abschnitt 4.1.3, in dem es allgemein um Fehlerbehandlung unter Verwendung der BPMN ging. Die dort präsentierten Lösungen lassen sich auch im Polling-Kontext zur Überwachung und Fehlerbehandlung anwenden.

Die beiden bisher betrachteten BPMN-Modelle zum Polling setzten die synchrone Abfrage des Zustands im Backend-System voraus. Stellt dieses allerdings den Service ebenfalls nur asynchron zur Verfügung, so muss das Modell, wie in Abbildung 78 gezeigt, verändert werden.

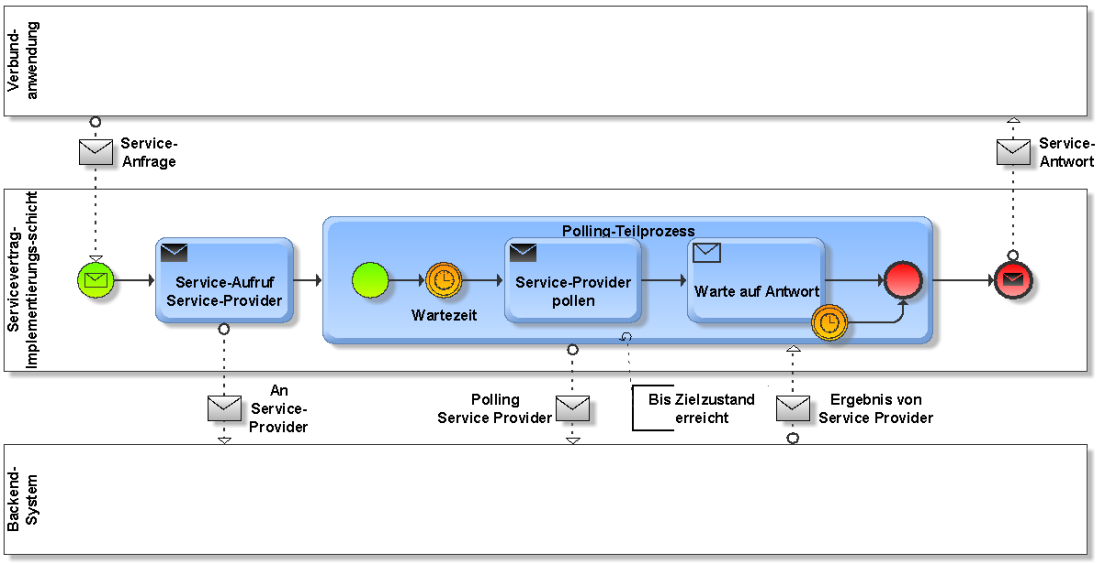



***Abbildung 78: Asynchrones Polling unter Verwendung eines Teilprozesses mit sequenzieller Schleife***

Auch in diesem Modell wird auf einen Polling-Teilprozess zurückgegriffen, der nun zwischen den separierten Aufrufen zum Polling-Versand und -Empfang unterscheidet. Zusätzlich wird die empfangende Aufgabe *Warte auf Antwort* mit einem angehefteten Timer-Zwischenereignis überwacht, um nicht unendlich auf die Antwort warten zu müssen. Stattdessen kann nach dem Ablauf des Timers unmittelbar mit einem erneuten Polling-Zyklus begonnen werden.

Composite Pattern übernehmen innerhalb der Architektur von Verbundanwendungen den wichtigen Aspekt der Vermittlung zwischen der Endanwendung und den beteiligten Geschäftsanwendungen. Sie zeichnen sich dabei durch eine aktive Rolle bei der Umsetzung des Servicevertrags aus. Dabei wurde bisher davon ausgegangen, dass die Antwort in einer zusammenhängenden Nachricht von dem Service-Provider zurückgeschickt werden konnte. Doch dies ist nicht immer der Fall. Aufgrund der Größe einer Nachricht muss diese unter Umständen aufgeteilt und die einzelnen Fragmente einzeln verschickt und anschließend auf Empfängerseite wieder zusammengefügt werden. Wie in solchen Fällen zu verfahren ist, verdeutlicht der nachfolgende Abschnitt über systemzentrische Pattern.

### 5.6.2 Systemzentrische Pattern

Systemzentrische Pattern zeichnen sich gegenüber den Composite Pattern dadurch aus, dass sie passiv auf das Eintreffen von Nachrichten warten, statt sie, wie im vorangegangenen Abschnitt gezeigt, aktiv durch Service-Aufrufe anzufordern. Im Mittelpunkt der Betrachtung stehen dabei die Pattern, bei denen sich die Antwortnachricht nicht aus einer Nachricht zusammensetzt, sondern aufgrund technischer oder betriebswirtschaftlicher Gründe in einzelnen Teilen geliefert wird. Ein technischer Grund kann die Gesamtgröße der Antwort sein, die nicht als Ganzes übertragbar ist. Ein betriebswirtschaftlicher Grund kann in der unterschiedlichen Behandlung einzelner Teile eines Serviceaufrufs liegen: werden beispielsweise die Positionen einer Bestellung aufgrund der georderten Produkte verschieden prozessiert, so resultieren daraus individuelle Nachrichten je Bestellposition. Der Serviceaufrufer erwartet aber eine in sich abgeschlossene Nachricht für seine gesamte Bestellung, damit diese als Ganzes weiterverarbeitet werden kann. In diesem Fall muss sich die Servicevertrag-Implementierungsschicht im Vergleich zu den Composition Pattern hinsichtlich ihres Verhaltens umstellen: sie wartet passiv auf die einzelnen Nachrichtenbestandteile, bis ein bestimmtes Endkriterium erfüllt und die Weiterverarbeitung angestoßen werden kann. Dieses Pattern ist in der Literatur auch unter dem Namen *Aggregator* bekannt (Hohpe & Woolf 2004, S. 268). Je nach Endkriterium sind nun wiederum unterschiedliche Szenarien denkbar, die im Folgenden betrachtet werden sollen. Dabei wird in den BPMN-Modellen jeweils nur das Einsammeln der einzelnen Nachrichtenfragmente und damit ausschließlich das Zusammenspiel zwischen den Backend-Systemen und der Implementierungsschicht dargestellt. Die Verbundanwendung taucht in den Modellen lediglich als Empfänger der Gesamtnachricht auf.

Im ersten Beispiel wird die Komplettierung durch eine Gesamtzahl zu erwartender Nachrichten bestimmt. Die Gesamtzahl wird dem Prozess dabei in einem bestimmten Feld der ersten Nachricht mitgeteilt. Das dazugehörige BPMN-Modell zeigt Abbildung 79.

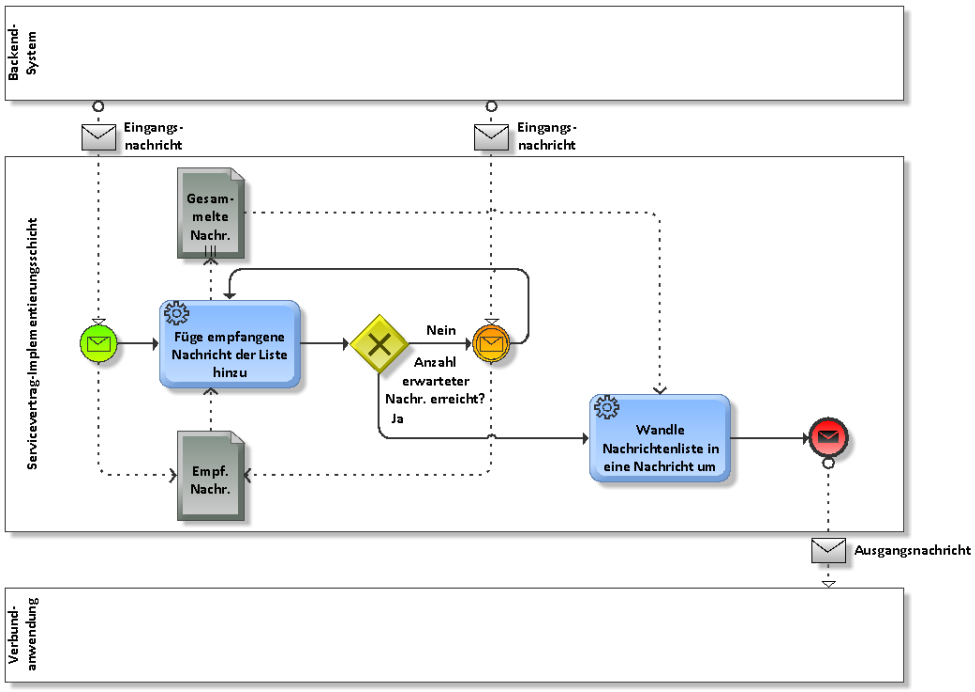



*Abbildung 79: Aggregator-Pattern mit explizit ausmodellierter Warteschleife*

Das Modell zeigt einmal mehr die explizite Ausmodellierung der Warteschleife, deren Ende durch das Eintreffen der letzten erwarteten Nachricht erreicht wird. Ebenfalls zu erkennen sind die expliziten Schritte zum Sammeln der Nachrichten in einem eigenen Datenobjekt, als auch die Umwandlung der resultierten Nachrichtenliste in eine Ausgangsnachricht. Die Eingangsnachrichten repräsentieren aufgrund derselben Namen identische Schnittstellen. Dadurch wird zum Ausdruck gebracht, dass der Prozess ausschließlich Nachrichten des selben Typs verarbeitet. Die Korrelationsbedingung für das nachrichtenbasierte Zwischenereignis ist abhängig vom umzusetzenden Szenario und könnte beispielsweise bei der Behandlung von Aufträgen die eindeutige Auftragsnummer sein, zu dem die Positionen gehören.

Als Alternative zur expliziten Ausmodellierung der Warteschleife kann auch ein sequenziell ausgeführter Teilprozess verwendet werden. Abbildung 80 zeigt das dazugehörige Modell.

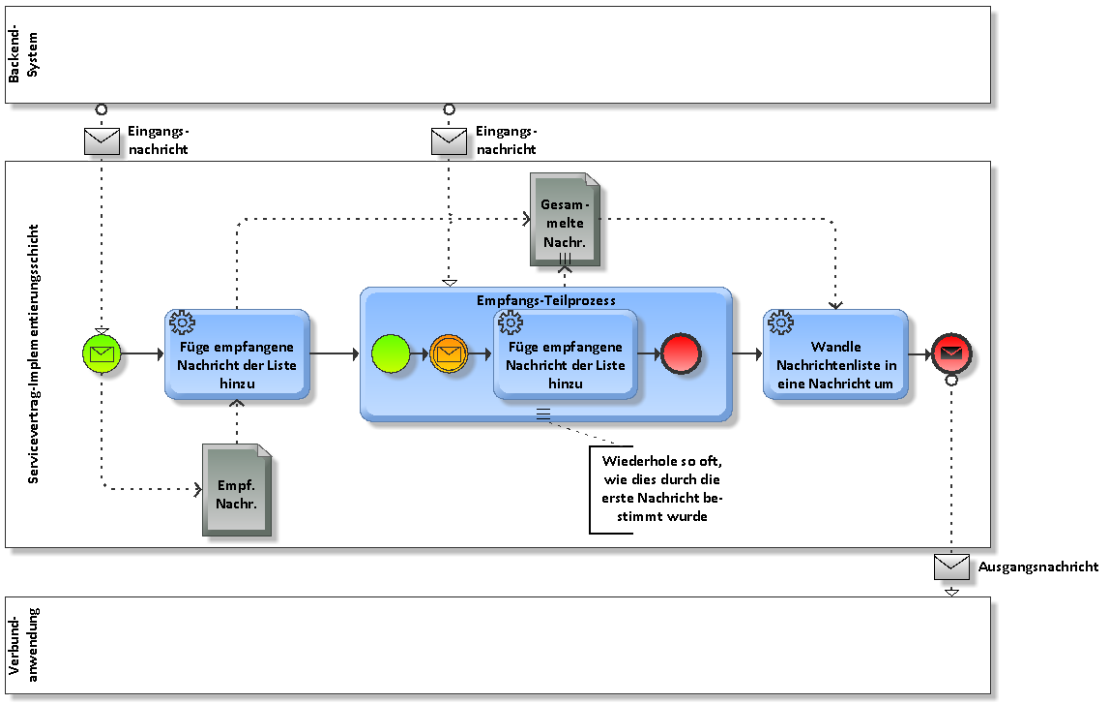



*Abbildung 80: Aggregator-Pattern mit sequenziell auszuführendem Teilprozess*

Der Empfangs-Teilprozess ist aufgrund der zur Laufzeit bekannten Anzahl der Schleifendurchläufe mit der sequenziellen Mehrfachausführung markiert. Der Vorteil der Verwendung eines Teilprozesses liegt in der Kombination mit angehefteten Ereignissen zur Zeit- und/oder Fehlerüberwachung. In beiden Modellen wurde sowohl das Sammeln der Nachrichten in einem eigenen Datenobjekt als auch die Erstellung der Ergebnisnachricht explizit mit Serviceaufgaben modelliert. Dies ist dann unnötig, wenn die zugrundeliegende Implementierung der BPMN-Spezifikation das Datenmapping in der Form unterstützt, wie dies im Implementierungskapitel mit SAP NetWeaver Business Process Management erläutert wurde. Dort wurde beim Verlassen einer Aufgabe der zur Aufgabe gehörende Dateninhalt in den Prozesskontext überführt, der sich aus den im Modell verwendeten Datenobjekten zusammensetzte. Umgekehrt wurde beim Erreichen einer Aufgabe der Inhalt des Prozesskontextes an den jeweiligen Schritt übergeben. Wird dieses Verhalten des Prozessmodells vorausgesetzt, so ergibt sich eine noch kompaktere Umsetzung. Sie ist in Abbildung 81 dargestellt.

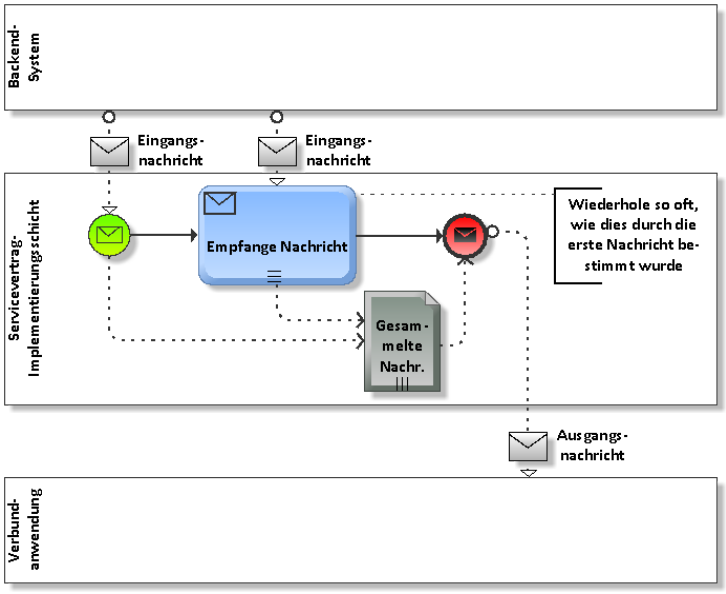



***Abbildung 81: Aggregator-Pattern mit sequenziell auszuführendem Teilprozess und implizitem Mapping***

Bereits das Start-Ereignis fügt bei seinem Verlassen durch ein implizites Output-Mapping die erste Nachricht zur Liste der gesammelten Nachrichten hinzu. Eine weitere Veränderung stellt die Verwendung der Empfangs-Aufgabe statt des nachrichtenbasierten Zwischenereignisses dar. Da auch ihr die Kennzeichnung der sequenziellen Mehrfachausführung mitgegeben werden kann, entfällt sowohl die explizite Schleifenmodellierung als auch der aufwändige Empfangs-Teilprozess. Zudem wird auch hier die empfangene Nachricht durch (implizites) Mapping dem *Gesammelte Nachrichten* Datenobjekt hinzugefügt. Schließlich wird beim Erreichen des sendenden Endereignisses die Umsetzung der Nachrichtenliste in die Datenstruktur der Ausgangsnachricht umgesetzt. Die verwendeten gerichteten Assoziationen erlauben in diesem Zusammenhang die detaillierte Modellierung des Datenflusses.

Allen drei Modellen gemein war der Abbruch der Schleife durch die in der ersten Nachricht vorgegebene Gesamtanzahl der zu erwartenden Nachrichten. Diese Voraussetzung ist jedoch nicht in allen Szenarien gegeben. So könnte in Prozessen, in denen es, wie bei Auktionen üblich, um Preisgebote geht, der Zeitfaktor das begrenzende Kriterium sein. In diesem Fall wird der Nachrichtenempfang durch den Ablauf eines Timers abgebrochen. Abbildung 82 veranschaulicht den Prozessablauf unter Verwendung eines Teilprozesses mit angeheftetem unterbrechenden Timer-Ereignis.

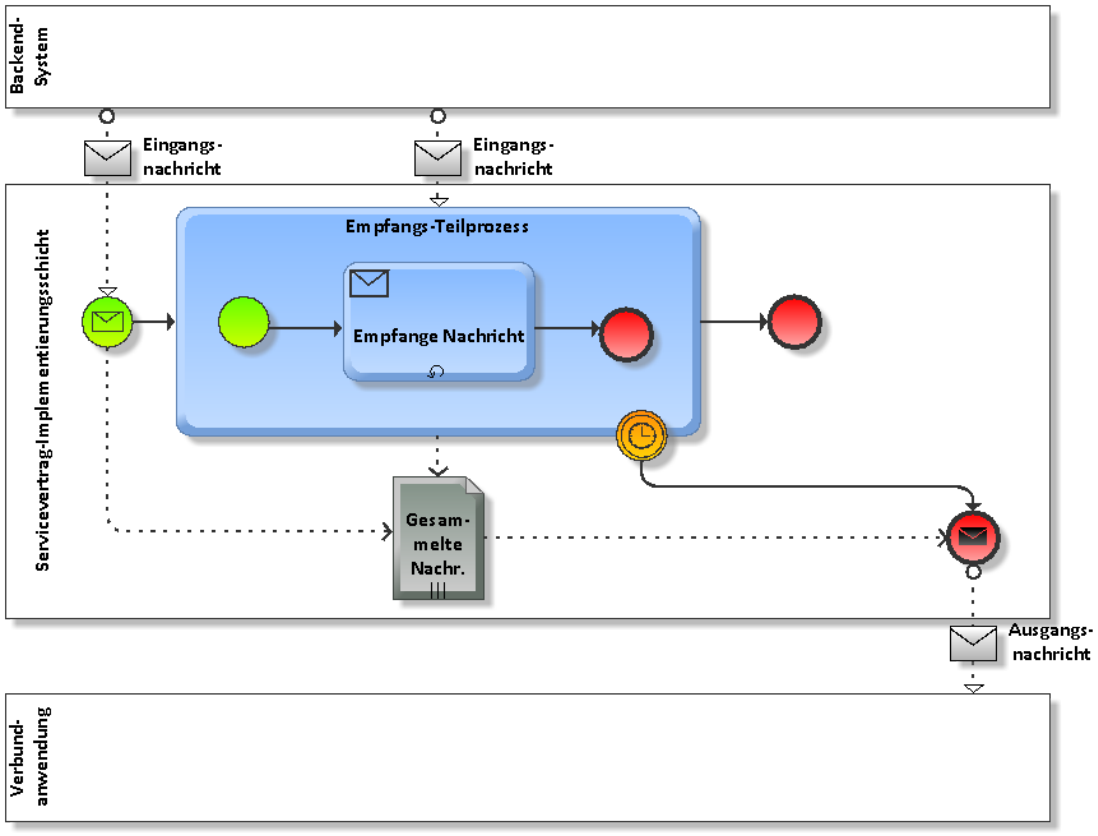



*Abbildung 82: Aggregator-Pattern mit Timer als Endekriterium für den Teilprozess*

Im Unterschied zu Abbildung 81 wurde die sequenzielle Mehrfachausführung der empfangenden Aufgabe durch eine einfache Schleife ersetzt, da die Anzahl der Schleifendurchläufe zu Beginn der Prozessausführung nicht bekannt ist. Zudem befindet sich die empfangende Aufgabe nun innerhalb eines Teilprozesses, der wiederum durch das angeheftete unterbrechende Timer-Zwischenereignis zeitüberwacht wird. Sobald die für den Empfang der Nachrichten vorgesehene Zeitspanne abgelaufen ist, wird die Empfangsschleife unterbrochen und die bis dahin gesammelten Informationen in einer Nachricht an die Verbundanwendung geschickt. Dieses Pattern ist natürlich nicht nur auf das Anheften des Timer-Ereignisses beschränkt. Es kann auch für andere Ereignisse wie Nachrichten, Bedingungen oder Signale verwendet werden und erhöht dadurch seine Wiederverwendbarkeit.

Bei Betrachtung des Modells stellt sich zwangsläufig die Frage, ob der Aufwand mit dem Teilprozess, der lediglich die empfangende Aufgabe enthält, wirklich betrieben werden muss, oder ob das Timer-Zwischenereignis nicht auch gleich an die Empfangsaufgabe geheftet werden kann, wie dies in Abbildung 83 dargestellt ist.

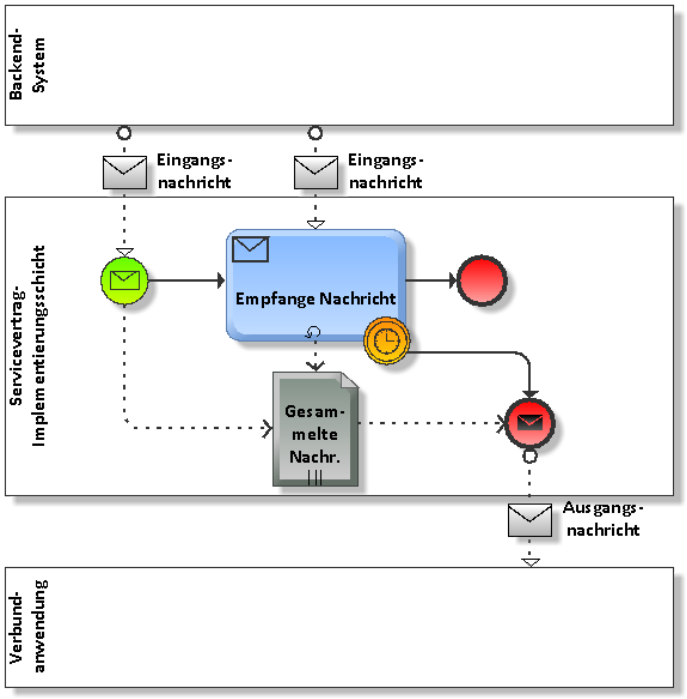



***Abbildung 83: Aggregator-Pattern mit Timer als Endekriterium für die Empfangs-Aufgabe***

Der Vergleich der beiden Modelle macht einmal mehr die Bedeutung des Gültigkeitsbereiches eines angehefteten Zwischenereignisses deutlich. Im ersten Fall (Abbildung 82) lässt sich der Gültigkeitsbereich leicht ermitteln: es wird der gesamte Teilprozess und damit die darin eingeschlossene Empfangsschleife überwacht. Während des Schleifendurchlaufs wird der Unterprozess nicht verlassen und damit die Zeitüberwachung mit jeder eingetroffenen Nachricht nicht neu gestartet. Das Modell ist semantisch korrekt. Doch wie verhält es sich bei dem Prozess aus Abbildung 83? Bezieht sich das angeheftete Timer-Zwischenereignis auf die gesamte Schleife oder lediglich auf den Empfang einer einzelnen Nachricht, so dass nach jedem Nachrichten-Empfang der Timer neu gesetzt wird? Die BPMN 2.0-Spezifikation (OMG 2010a, S. 472) äußert sich diesbezüglich wie folgt:

„*The Loop Activity is a type of Activity that acts as a* ***wrapper*** *for an inner Activity that can be executed multiple times in sequence.*

***Operational semantics****: Attributes can be set to determine the behavior. The Loop Activity executes the inner Activity as long as the loopCondition evaluates to true.*“

Tatsächlich *enthält* die Schleifen-Aktivität die Empfangsaufgabe und folglich bezieht sich das Timer-Zwischenereignis ebenfalls auf die Schleife und nicht auf die Einzelaufgabe. Durch beide Modelle wird also dasselbe Verhalten definiert.

Ein weiteres mögliches Endkriterium für die Sammelschleife ist der Empfang einer dedizierten Endenachricht. Für diesen Fall bietet BPMN mehrere interessante Modellierungsmöglichkeiten an. Abbildung 84 zeigt eine erste Lösung in Form einer expliziten Schleife.

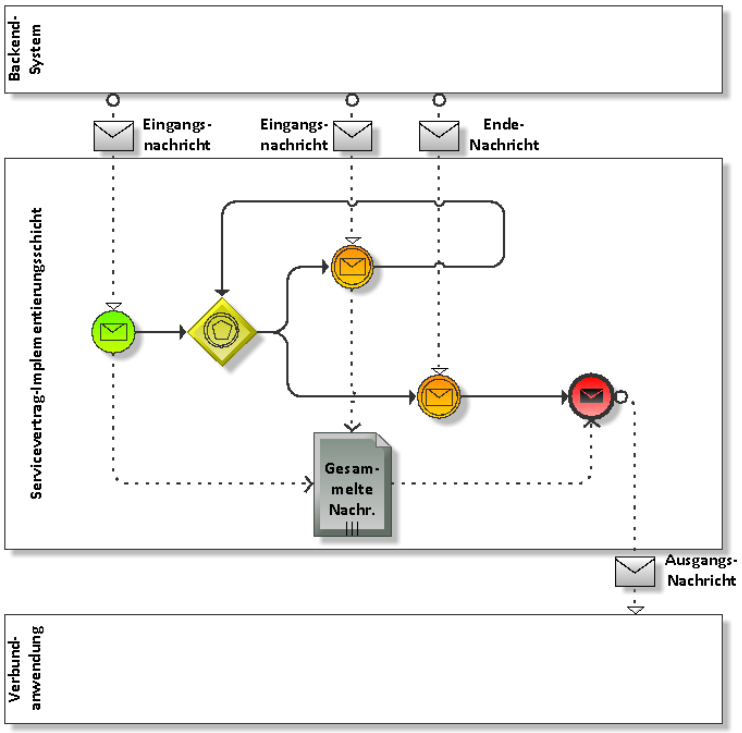



***Abbildung 84: Aggregator-Pattern mit dedizierter Endenachricht als Endekriterium modelliert mit ereignisbasiertem Gateway***

Nach Start des Prozesses durch die Eingangsnachricht wartet dieser anschließend am ereignisbasierten Gateway auf das mögliche Eintreffen zweier unterschiedlicher Nachrichtentypen, die im Modell durch verschiedene Namen dargestellt sind. Weitere Eingangsnachrichten werden wie gewohnt in der Schleife gesammelt, während das Eintreffen der *Ende-Nachricht* für die Beendigung der Empfangsschleife und damit den Versand der Sammelnachricht an die Verbundanwendung sorgt.

Statt der expliziten Ausmodellierung der Schleife kann auch in diesem Fall der Teilprozess-Ansatz wiederverwendet werden, so wie er im Zusammenhang mit dem angehefteten Timer-Zwischenereignis in Abbildung 82 dargestellt wurde. In der Abbildung müsste dann lediglich das angeheftete Timer-Zwischenereignis durch ein nachrichtenbasiertes Zwischenereignis ausgetauscht werden. Folglich wird der Teil-

prozess, der lediglich die Empfangsaufgabe mit Standard-Schleifenmarkierung enthält, statt durch das Ablaufen des Timers durch den Erhalt der Endenachricht unterbrochen. Die restliche Ablauflogik bleibt von dieser Änderung ansonsten unberührt.

Eine weitere interessante Alternative zeigt Abbildung 85.

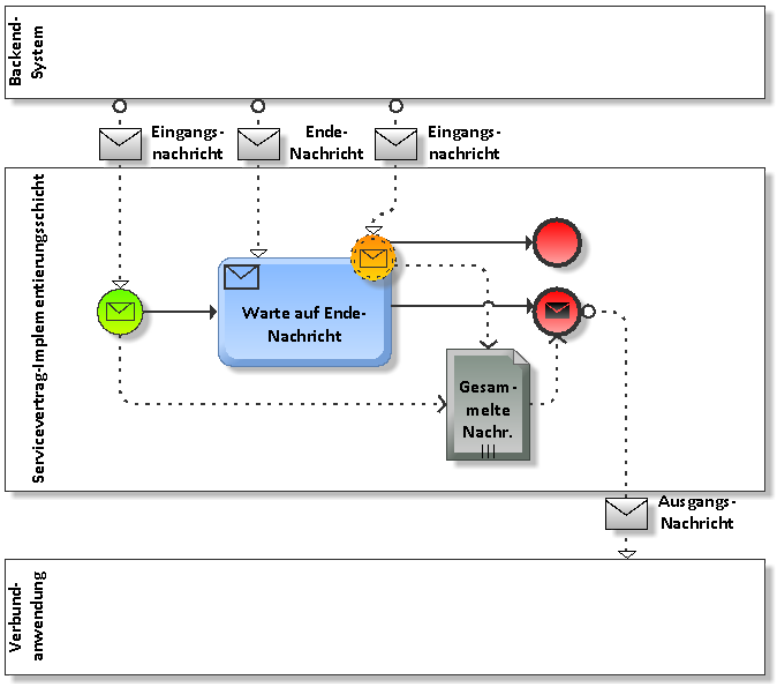



***Abbildung 85: Aggregator-Pattern mit dedizierter Endenachricht als Endekriterium modelliert mit nicht-unterbrechendem Nachrichtenereignis***

In diesem Modell wird nach dem Erhalt der ersten *Eingangsnachricht* an der empfangenden Aufgabe auf das Eintreffen der *Ende-Nachricht* (!) gewartet. Dieser Aufgabe wurde allerdings ergänzend ein nichtunterbrechendes nachrichtenbasiertes Zwischenereignis angeheftet, das auf ankommende *Eingangsnachrichten* reagiert. Dadurch wird die parallele Verarbeitung mehrerer gleichzeitig eintreffender Nachrichten ermöglicht, unterstützt also die hochperformante Parallelverarbeitung von Nachrichten. Die einzelnen parallelen Pfade enden in einem untypisierten Ende-Ereignis und beenden daher den Gesamtprozess nicht. Erst nach Empfang der *Ende-Nachricht* erreicht der Prozess das nachrichtenbasierte Ende-Ereignis und schließt den Prozessfluss ab.

Zum Abschluss der Diskussion von Pattern zum Sammeln von Nachrichten sei noch auf zwei Sonderfälle hingewiesen. In allen bisher betrachteten Fällen wurde der

Sammelprozess durch exakt eine wohldefinierte Startnachricht initiiert. Es sind hingegen auch Fälle denkbar, bei denen in verschiedenen Systemen Aktivitäten angestoßen werden und es nicht vorhersehbar ist, welches System als Erstes antwortet. Erschwerend kommt hinzu, dass die Systeme mit unterschiedlichen Schnittstellen antworten. Der Prozess muss also auf den Empfang verschiedenster Nachrichten in unbekannter Reihenfolge vorbereitet sein. Ein typisches Szenario für dieses Problem ist das Zusammentragen von Kundeninformationen aus unterschiedlichsten Systemen. Klassische Kunden-Stammdaten sind beispielsweise zu ergänzen um Konteninformationen, Kreditwürdigkeit, Einkaufsverhalten, die letzten Aufträge usw. Diese Daten können unabhängig voneinander bei den Backend-Systemen angefordert werden. Allein die Reihenfolge der Antworten ist ungewiss. Eine mögliche Lösung dieses Problems veranschaulicht Abbildung 86.

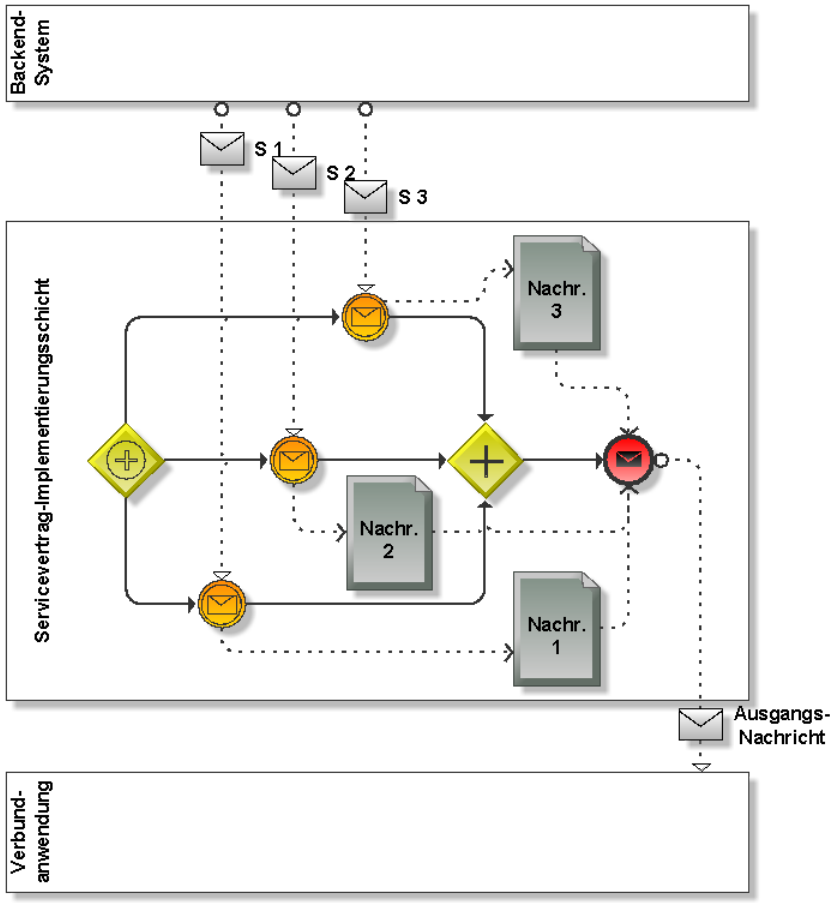



***Abbildung 86: Aggregator-Pattern bei unbekannter Reihenfolge des Nachrichteneingangs***

Bezeichnend für dieses Modell ist die Verwendung des parallelen ereignisbasierten Gateways zur Prozessinstanziierung gleich zu Beginn des Sequenzflusses. Durch dieses Gateway wird der Start des Prozesses veranlasst, sobald *eines* der dem Gateway nachfolgenden Ereignisse eintritt. Folglich ist das Eintreffen einer der drei Nachrichten, die durch die unterschiedlichen Schnittstellen S1, S2 und S3 repräsentiert werden, für den Start ausschlaggebend. Nach dem Start wird auf die noch ausstehenden Nachrichten gewartet und erst nach deren Empfang, gewährleistet durch das parallele Gateway bei der Zusammenführung der parallelen Pfade, wird der Sequenzfluss mit dem Versand des gesammelten Ergebnisses durch das nachrichtenbasierte Ende-Ereignis fortgesetzt.

Zum Abschluss soll noch eine leicht abgeänderte Variante des zuvor beschriebenen Szenarios betrachtet werden: im vorangegangenen Beispiel wurde die Zusammenführung über das parallele Gateway gesteuert. Es wurde folglich auf alle ausstehenden Nachrichten gewartet. Dies ist jedoch nicht in allen Situationen notwendig. Daten werden unter Umständen redundant in verschiedensten Backend-Systemen gehalten. Man kann diese Systeme aber unabhängig voneinander ansprechen und gleichartige Daten von ihnen anfordern. Je nachdem wie die aktuelle Last in den Geschäftssystemen verteilt ist, werden die Systeme mit unterschiedlichen Antwortzeiten reagieren. Liefert dabei ein System A bereits Daten, die auch ein System B liefern würde, macht es keinen Sinn, auch noch auf dessen Antwort zu warten, da die benötigten Daten bereits vorliegen. Andererseits macht es durchaus Sinn, mehrere Systeme für dieselben Daten zu kontaktieren, da dadurch die Robustheit und Gesamtverfügbarkeit der Lösung erhöht und die Antwortzeit für die Verbundanwendung reduziert wird. Allerdings kann in einem solchen Fall nicht auf das parallele Gateway für die Zusammenführung der parallelen Stränge zurückgegriffen werden. Auch das inklusive und exklusive Gateway scheiden aus. Stattdessen sieht die BPMN für solche Fälle das komplexe Gateway vor, in dem beliebige Bedingungen für die Zusammenführung definiert werden können. Abbildung 87 veranschaulicht diesen Sachverhalt.

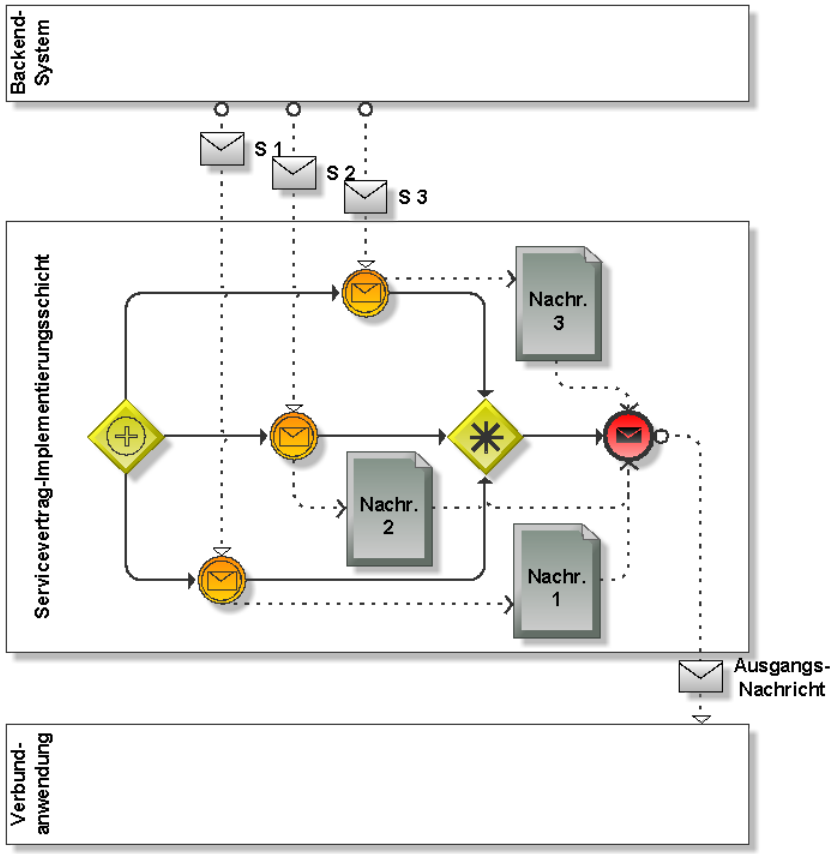



***Abbildung 87: Aggregator-Pattern unter Verwendung des komplexen Gateways***

Allerdings ist dem Modell nicht direkt zu entnehmen, welche Bedingungen dem Gateway konkret zugewiesen wurden. Diese Informationen könnten in Form von Kommentaren ergänzt werden. Für das obige Beispiel könnte dies bedeuten, dass der Prozessfluss fortgesetzt wird, sobald entweder die Nachrichten S1 und S3 oder S2 und S3 eingetroffen sind. Dabei wird allerdings vorausgesetzt, dass S1 und S2 gleichartige Daten darstellen.

Für ausführbare Prozesse müssen die Bedingungen in etwa so formuliert werden, wie dies im Implementierungsbeispiel aus Kapitel 4 für die Pfade des Gateways oder der Korrelationsbedingung für das nachrichtenbasierte Zwischenereignis gezeigt wurde. Die Verwendung des komplexen Gateways eröffnet dem Modellierer für derartige Fälle zusätzliche Freiheitsgrade.

Durch die Verwendung von Pattern können die im Projekt-Alltag immer wiederkehrenden Probleme mit bewährten Lösungen angegangen werden. In diesem Unter-

kapitel wurden anhand konkreter Beispiele solche Lösungen für die Servicevertrag-Implementierungsschicht herausgearbeitet und verschiedenste Alternativen diskutiert. Bei der Umsetzung der Prozessfragmente  konnte einmal mehr die BPMN aufgrund ihrer Ausdrucksstärke behilflich sein und ihren Einsatz auch in eher technischen Prozessen unter Beweis stellen. Wie die Flexibilität durch den Einsatz von Regelwerken zusätzlich gesteigert werden kann, betrachtet Abschnitt 5.7 (Flexibilitätsgewinn durch die Verwendung von Regelwerken in Kombination mit analytischen Anwendungen). Im nun folgenden Abschnitt ist allerdings zuvor ein Erweiterungsvorschlag für BPMN zur Modellierung von Integrationsprozessen Gegenstand der Diskussion.

### 5.6.3 *Erweiterungsvorschlag für BPMN zur dedizierten Modellierung von Integrationsprozessen*

In den beiden vorangegangenen Abschnitten 5.6.1 und 5.6.2 wurden eine Vielzahl von Pattern insbesondere für die Servicevertrag-Implementierungsschicht diskutiert sowie deren Umsetzung mit BPMN gezeigt. Offensichtlich eignet sich BPMN aufgrund der Vielzahl technisch-orientierter Komponenten sehr gut zur Implementierung von Integrationsprozessen. Es stellt sich zwangsläufig die Frage, ob es Sinn macht, Enterprise Integration Pattern, wie im Buch von Hohpe und Woolf diskutiert (Hohpe & Woolf 2004), zum Bestandteil von BPMN selbst zu machen? Hohpe und Woolf stellen in ihrem Buch eine eigene Notation vor (Hohpe & Woolf 2004, S. xliv der Introduction), die auf einfachen Symbolen basiert, wobei jedes Symbol ein konkretes Enterprise Integration Pattern repräsentiert. Das in Abbildung 88 dargestellte Symbol stellt beispielsweise das Aggregator-Pattern dar (Hohpe & Woolf 2004, S. 268).

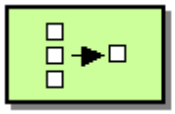


***Abbildung 88: Symbol für das Aggregator-Integrationspattern (aus Hohpe & Woolf 2004, S. 268)***

Eine Aneinanderreihung derartiger Symbole drückt demnach eine Verarbeitungskette (in der Literatur oftmals auch mit Pipeline oder Route bezeichnet) für eine Nachricht innerhalb einer Integrationslösung aus.

Die vorgeschlagene Notation hat sich mittlerweile bei Integrations-Entwicklern etabliert. Es gibt sogar eine Unterstützung der Symbole für das Microsoft-Produkt Visio (siehe EIP 2011), so dass sich Lösungen von Integrationsproblemen mit Hilfe der empfohlenen Notation recht prägnant formulieren lassen. Dessen ungeachtet bleibt auch diese Notation hinsichtlich der Semantik zur Laufzeit wichtige Details schuldig. Wird beispielsweise das Symbol für das Aggregator-Integrationspattern aus Abbildung 88 verwendet, so weiß der Leser zwar, dass zu diesem Zeitpunkt Nachrichten aggregiert werden sollen. Wie diese Aggregation allerdings konkret umzusetzen ist, woher die Nachrichten kommen und welches Ergebnis der Aggregator letztendlich produziert, bleibt dem Betrachter verborgen. So ist der Darstellung nicht zu entnehmen, wann die Aggregation beendet werden soll, ob also, wie in Hohpe & Woolf 2004, S. 272/273 vorgeschlagen, eine vorher ermittelte konkrete Nachrichtenanzahl abzuwarten ist, ob die Aggregation automatisch nach einer bestimmten Zeitspanne beendet wird oder ob ein externes Ereignis für dessen Beendigung sorgt. Daher stellt sich zwangsläufig die Frage, ob gegebenenfalls durch eine Kombination von BPMN und der Pattern-Symbolik eine aussagekräftigere Lösung entsteht, die dem Betrachter eines derartigen Modells wesentlich mehr Informationen zur Verfügung stellt, als dies allein auf Basis der Pattern-Ikonen möglich wäre. Schließlich stellt eine Verkettung von Integrationspattern auch eine Art von Prozess dar, der von den bereits hervorgehobenen Vorteilen der BPMN hinsichtlich der Ausdrucksfähigkeiten insbesondere bei der Ereignisverarbeitung zusätzlich profitieren könnte. Der nun folgende Abschnitt nimmt sich daher dieser Fragestellung an.

#### 5.6.3.1 Spracherweiterungen für BPMN um Integrationspattern zur prägnanten Darstellung von Integrationsprozessen

In dem Buch von Hohpe & Woolf werden nicht weniger als 65 Integrationspattern diskutiert. Allerdings sind nicht alle für Integrationsprozesse relevant. So werden die Patterns in sechs Kategorien eingeteilt (Hohpe & Woolf 2004, S. xlvii der Introduction):

- Nachrichtenkanäle (Message Channel) – über welche Wege können Anwendungen und Systeme mit Integrationslösungen kommunizieren?
- Konstruktion von Nachrichten (Message Construction) – wie können Nachrichten von ihrem Aufbau her aussehen?

- Routing von Nachrichten (Message Routing) – wie sieht die Nachrichtenverarbeitung innerhalb einer Integrationslösung aus und wie wird ermittelt, wohin eine Nachricht weitergeleitet werden muss?
- Nachrichtentransformation (Message Transformation) – wie kann zwischen den verschiedenen Datenformaten der beteiligten Systeme und Anwendungen vermittelt werden?
- Endpunkte (Message Endpoints) – wie können Anwendungen und Systeme, die ursprünglich nicht für ihre Teilnahme an Integrationsszenarien konzipiert wurden, dazu befähigt werden?
- Systemmanagement (Systems Management) – wie kann der korrekte Betrieb von implementierten Integrationsszenarien sichergestellt und überwacht werden?

BPMN konzentriert sich von seiner Ausrichtung her stark auf die Sequenzsteuerung von Prozessabläufen sowie dem damit verbundenen Nachrichtenfluss sowohl innerhalb des Prozesses selbst als auch mit externen Partnern in Form von Kollaborationsdiagrammen. Daher bieten sich aus obiger Einteilung insbesondere die Kategorien *Message Routing* und *Message Transformation* an. Dabei wird nun im Folgenden untersucht, wie die Integrationspattern in die BPMN integriert werden können. Im Wesentlichen wird eine Erweiterung der Aufgabentypen vorgeschlagen, wobei die linke obere Ecke der Aufgabe zur besseren Erkennbarkeit mit dem Symbol des dazugehörigen Integrationspatterns versehen wird. Es ist zudem zu überlegen, ob das Hinzufügen einer neuen Task in den Prozessfluss bereits ausreichend ist, oder ob weitere Details wie Beziehungen zu anderen Elementen in Form von Assoziationen die Verständlichkeit des Patterns und damit des Gesamtprozesses erhöhen. Bei einigen wenigen sehr komplexen Pattern, wie beispielsweise dem Aggregator-Pattern, muss des Weiteren geklärt werden, ob dessen Implementierung nicht noch zusätzlich in Form eines Teilprozesses detaillierter ausgeführt werden sollte. Schlussendlich muss auch die Ausführbarkeit der neuen Aufgaben Rechnung getragen werden: welche ergänzenden Informationen (Metadaten) sind der neuen Aufgabe hinzuzufügen, damit sie in einer Prozess-Engine ausgeführt werden kann? Zur Beantwortung dieser Frage wird auf das Open Source Projekt *Apache Camel* (siehe Camel 2011) als Referenzimplementierung für die von Hohpe & Woolf diskutierten Enterprise Integration Pattern (EIP) zurück-

gegriffen und verglichen, welche Parametrisierung dort für ein konkretes EIP vorgesehen ist. Als Quellen dienen hierzu das Buch von Claus Ibsen und Jonathan Anstey mit dem Titel *Camel in Action* (Ibsen & Anstey 2011) und die Online-Dokumentation der Firma FuseSource, die einen grafischen Camel-Editor basierend auf den EIP-Symbolen anbietet (FuseSource 2011a). Dazu werden im Folgenden die einzelnen Pattern ohne Einhaltung einer bestimmten Reihenfolge und ohne Anspruch auf Vollständigkeit diskutiert. Die nun folgenden Ausführungen über die verschiedenen Pattern erheben auch nicht den Anspruch, die Pattern formal bis ins letzte Detail spezifizieren zu wollen. Ziel ist es, die wesentlichen Ideen, die den Pattern zugrunde liegen, zu erfassen. Eine weitergehende Ausarbeitung der Spracherweiterung in der Form, dass sie offiziell als BPMN-Erweiterung bei der OMG eingereicht werden könnte, muss daher weitergehenden Forschungsarbeiten überlassen bleiben.

#### 5.6.3.2 Aggregator-Pattern

Das Aggregator-Pattern stellt gleich zu Beginn eines der komplexesten Pattern überhaupt dar. Im Rahmen der systemzentrischen Pattern im Kapitel 5.6.2 wurde dieses Pattern bereits intensivst diskutiert. Zur Erinnerung: das Pattern fasst mehrere zusammengehörige Nachrichten zu einer einzelnen neuen Nachricht zusammen. Dabei ergaben die Untersuchungen aus Kapitel 5.6.2, dass aufgrund der vielfältigen Parametrisierungsmöglichkeiten unterschiedlichste BPMN-Modelle zur Implementierung seines Verhaltens notwendig wurden. So müssen letztendlich drei wesentliche Eigenschaften bei seiner Umsetzung berücksichtigt werden (aus Hohpe & Woolf 2004, S. 270):

1. Korrelation: welche eingehenden Nachrichten gehören zusammen?
2. Abbruch-Bedingung: wann wird das Warten auf weitere eingehende Nachrichten beendet?
3. Aggregierungs-Algorithmus: wie werden die verschiedenen Nachrichten zu der einen neuen Nachricht verarbeitet?

Von daher bietet es sich an, dieses Pattern nicht als eine einzige neue Aufgabe darzustellen, sondern stattdessen einen ganzen Teilprozess vorzusehen, der im zusammengeklappten Zustand lediglich das Symbol aus Abbildung 88 in der linken oberen Ecke trägt, im aufgeklappten Zustand jedoch die Details der Implementierung berücksichtigt und die verschiedensten Umsetzungsoptionen (wie in Kapitel 5.6.2 bespro-

chen) grafisch darlegt. Dazu ist in Abbildung 89 das Aggregator-Pattern in der kompakten Teilprozess-Darstellung und in Abbildung 90 mit seinen Details dargestellt.

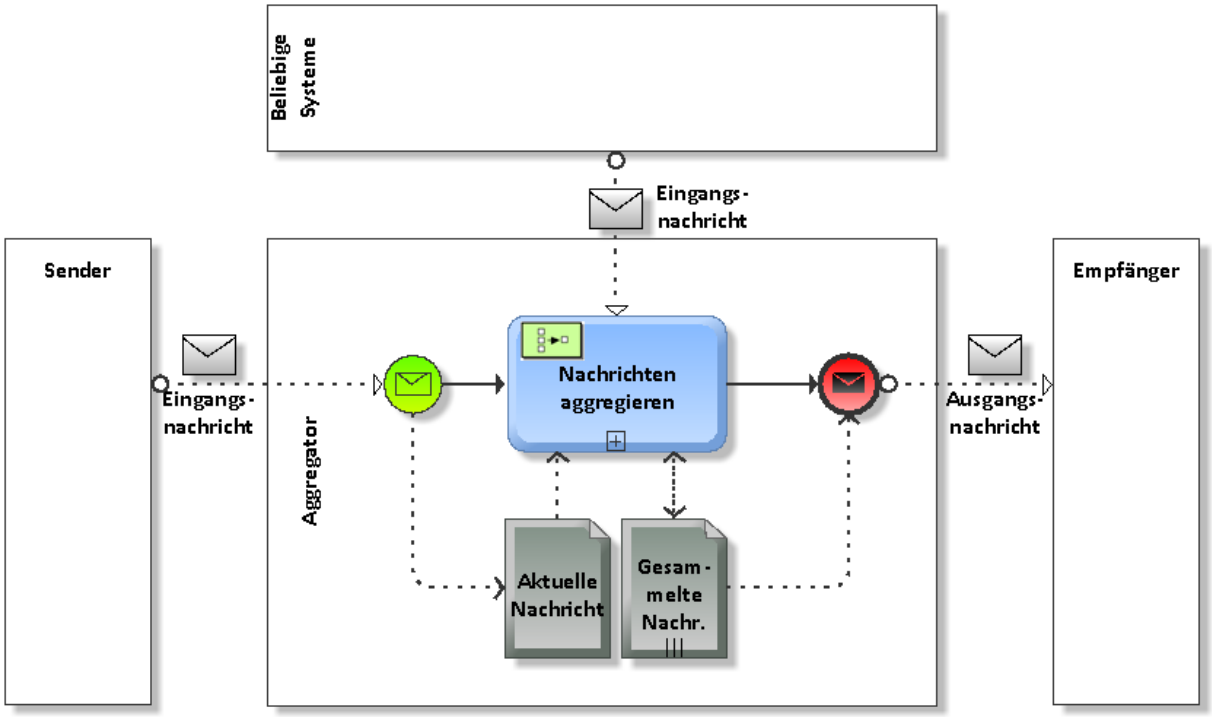



*Abbildung 89: Aggregator-Pattern in kompakter Darstellung*

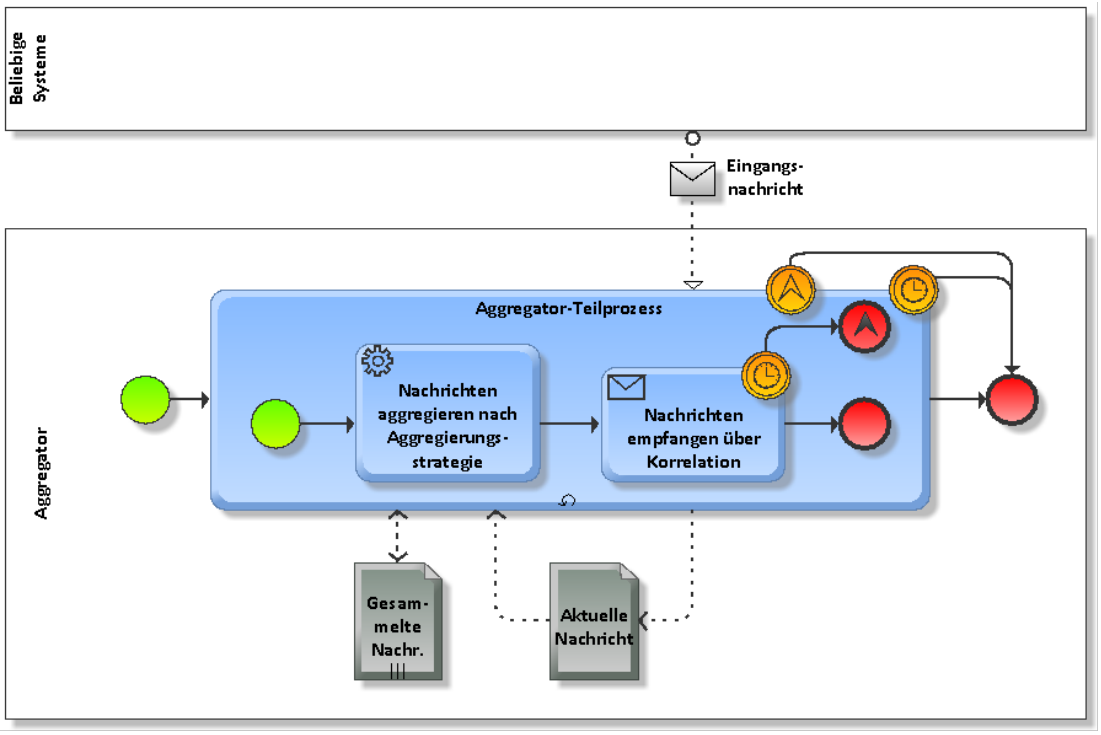



*Abbildung 90: Details des Aggregator-Teilprozesses*

In Abbildung 89 ist zunächst der Eingang der ersten Nachricht dargestellt, die letztendlich die Verarbeitung instanziiert. Die eingegangene Nachricht wird in einem Datenobjekt *Aktuelle Nachricht* hinterlegt, das auch gleichzeitig zur Kommunikation mit dem Unterprozess Verwendung findet (in beiden Modellen ist das Datenobjekt identisch benannt). In Abbildung 90 wird die neue Nachricht zuerst gemäß der Aggregierungsstrategie zur Liste der gesammelten Nachrichten hinzugefügt. Anschließend wird in einer Empfangsaufgabe auf weitere Nachrichten gewartet und diese nach deren Erhalt aufgrund der erfüllten Korrelationsbedingung erneut über die Aggregierungsstrategie der Sammelliste hinzugefügt. Die eingangs erwähnten fundamentalen Konfigurationseigenschaften des Aggregator-Patterns (Korrelation, Abbruch-Bedingung sowie Aggregierungsalgorithmus) lassen sich nun in das Prozessmodell aus Abbildung 90 wie folgt einbringen:

- Die Korrelation wird als Eigenschaft der Empfangsaufgabe aus Abbildung 90 definiert. Diese kann beispielsweise über ein Script formuliert werden wie dies bei der Implementierung der Grundarchitektur einer Verbundanwendung in Abschnitt 4.2.3.6 gezeigt wurde.
- Der Aggregierungsalgorithmus, der bestimmt, wie eingegangene Nachrichten zu verschmelzen sind, ist in Abbildung 90 der Aufgabe *Nachrichten aggregieren nach Aggregierungsstrategie* zugeordnet und verweist dabei auf eine Komponente, die konkret die Einarbeitung der neuen Daten in die bereits aufgebaute Sammelnachricht durchführt. Dies könnte wiederum ein Script sein. Alternativ ist auch der Einsatz von Java Beans denkbar.
- Bleibt die Umsetzung der Abbruchbedingung. In Ibsen & Anstey 2011, S. 244 werden vier wesentliche Beendigungskriterien genannt, die im Modell ebenfalls berücksichtigt werden können:

    1. Beendigung bei Erreichen einer bestimmten Gesamtnachrichtenanzahl (Completion Size): diese Bedingung kann als Eigenschaft des Aggregator-Teilprozesses aus Abbildung 90 hinterlegt werden und damit als Kriterium für die Standardschleife dienen.
    2. Beendigung, nachdem für eine gewisse Zeit keine Nachricht mehr eingegangen ist (Completion Timeout): diese Bedingung wird in Abbil-

dung 90 durch das angeheftete unterbrechende Timer-Zwischenereignis an der Empfangsaufgabe dargestellt.

3. Beendigung zu einem bestimmten Zeitpunkt (Completion Interval): im Modell aus Abbildung 90 wird dafür das angeheftete unterbrechende Timer-Zwischenereignis am Aggregator-Teilprozess verwendet. Sobald die Zielzeit für die Gesamtverarbeitung erreicht ist, wird die Schleife des Teilprozesses unterbrochen und damit die gesamte Nachrichtenbehandlung abgeschlossen.
4. Beendigung einer Bedingung, die sich aus dem Nachrichteninhalt ergibt (Completion Predicate): diese inhaltsbezogene Überwachung kann wiederum den Attributen des Aggregator-Teilprozesses aus Abbildung 90 hinzugefügt werden. Formulierbar ist eine solche Bedingung wieder über ein Skript oder, da es sich auf den Inhalt der Nachricht bezieht, auch über einen XPath-Ausdruck. Auch sie dient gleichzeitig als Abbruchkriterium für die Standardschleife.

Die Beendigungsbedingungen lassen sich auf diese Weise beliebig kombinieren, so wie es das Pattern aus Hohpe & Woolf 2004 auch vorsieht.

Wird das Aggregator-Pattern in einer Verarbeitungskette eingesetzt, die weitere Enterprise Integration Patterns umfasst und bei der das Aggregator-Pattern **nicht** als erste Aufgabe nach dem Nachrichtenempfang eingesetzt wird, so kann die Implementierung aus Abbildung 90 nicht davon ausgehen, dass das Datenobjekt A*ktuelle Nachricht* bereits gefüllt ist. Der Sequenzfluss aus Abbildung 90 muss folglich mit dem Nachrichtenempfang beginnen, ehe mit der Aggregierungsstrategie fortgefahren werden kann. Das Modell aus Abbildung 91 berücksichtigt diesen Sachverhalt.

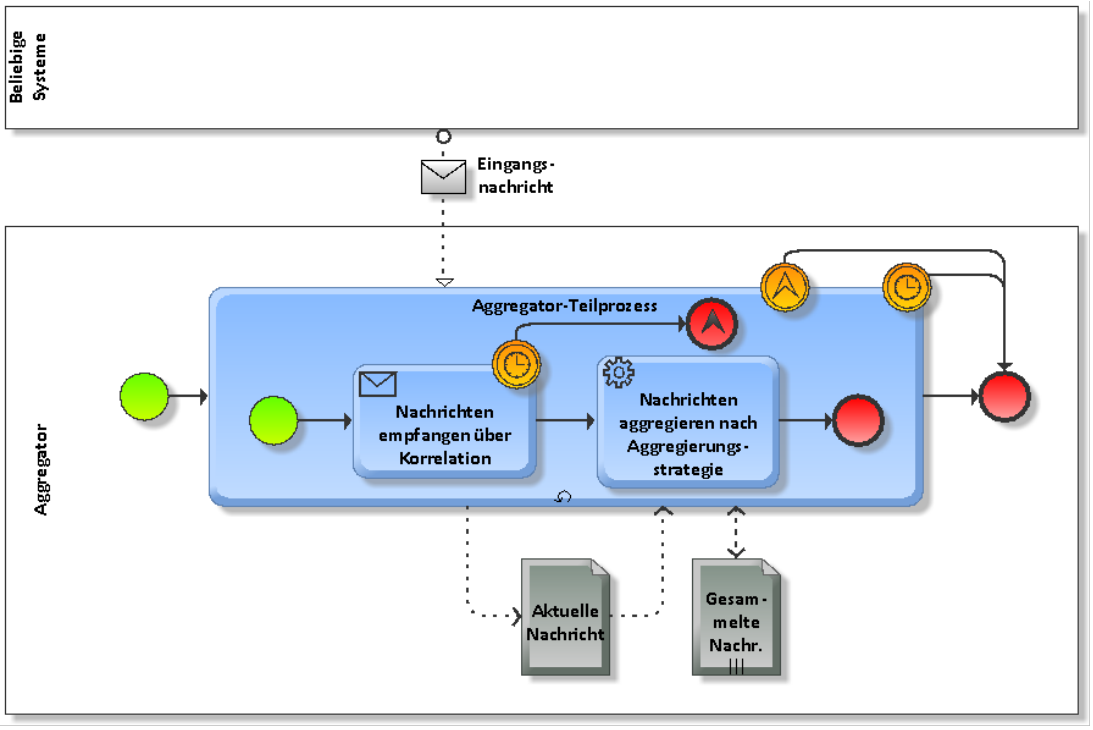



*Abbildung 91: Details der Aggregator-Implementierung bei Verwendung des Aggregators innerhalb einer Verarbeitungskette*

#### 5.6.3.3 Content Enricher-Pattern

Ziel des Content Enricher-Patterns ist es, eingehende Daten so um Informationen zu ergänzen, dass die Nachricht vom Zielsystem verarbeitet werden kann (Hohpe & Woolf 2004, S. 336). Oftmals ist der Sender hingegen nicht in der Lage, sämtliche der im Zielsystem benötigten Daten zu liefern. In diesem Fall sorgt der Content Enricher durch einen synchronen Aufruf dafür, dass die notwendigen Daten ermittelt werden. Die dazugehörige neue BPMN-Aufgabe ist in Abbildung 92 zu sehen.

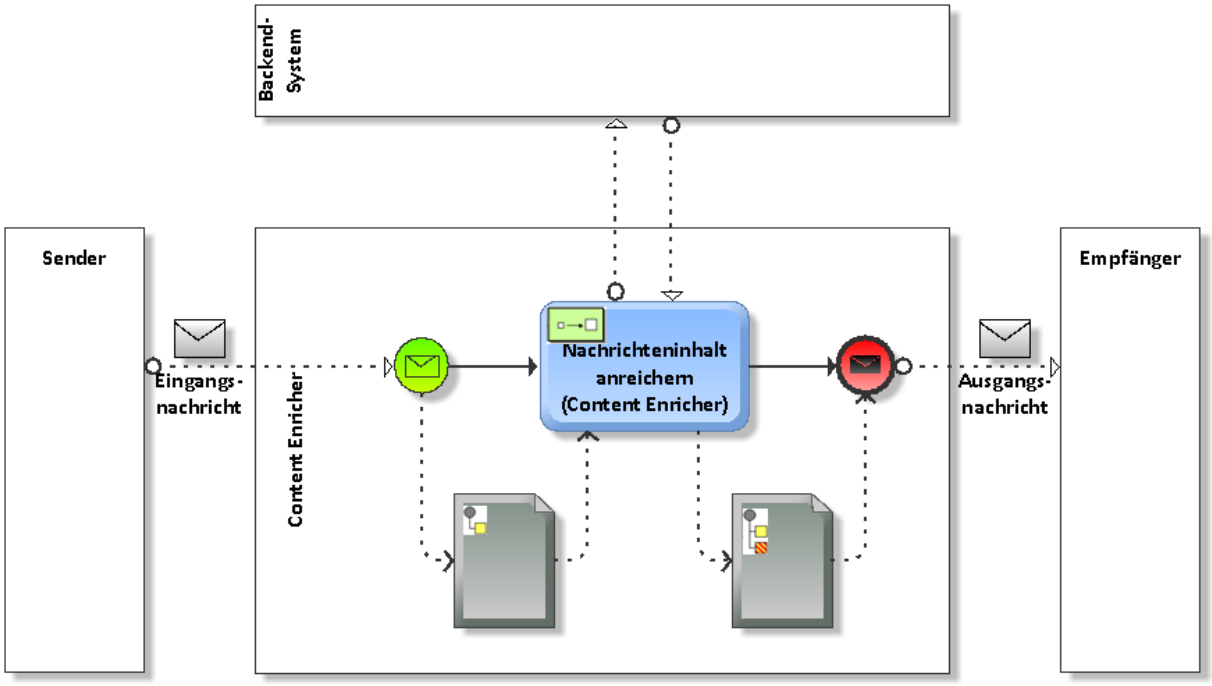



*Abbildung 92: Content Enricher-Pattern*

Der Content Enricher bekommt die Ursprungsnachricht als Eingabeparameter mitgeliefert, bedient sich anschließend eines weiteren Systems zur Beschaffung der fehlenden Informationen und reichert die Originalnachricht damit an. Visuell ist dies durch ebenfalls neue Symbole in den linken oberen Ecken der Datenobjekte zu erkennen, die auch aus der Symbolsprache der Integrationspattern stammen (Hohpe & Woolf 2004, S. xliv der Introduction). Es ist zu erkennen, wie das ursprüngliche Datenobjekt, versehen mit lediglich einem Quadrat, nach dem Content-Enricher-Durchlauf um ein weiteres Quadrat ergänzt wurde.

Damit die neue Content-Enricher-Aufgabe auch ausführbar ist, müssen die folgenden beiden Informationen spezifiziert werden (FuseSource 2011b):

- Die Adresse zur Quelle, von der die zusätzlichen Informationen angefordert werden. In obigem BPMN-Modell könnte diese Information beispielsweise dem Backend-System-Pool zugeordnet werden, mit dem der Content Enricher kommuniziert.
- Aggregierungsstrategie: ähnlich wie beim Aggregator-Pattern muss der Ausführungs-Engine mitgeteilt werden, wie die angeforderten Daten schließlich in die Originalnachricht eingearbeitet werden soll. Die Funktio-

nalität selbst kann über eine Java Bean oder ein Script erfüllt werden. Der Verweis auf Bean bzw. Script ist hingegen Bestandteil der Eigenschaften der Content-Enricher-Aufgabe selbst.

Derart konfiguriert kann der Content Enricher zur Laufzeit seine Aufgabe erfüllen.

#### 5.6.3.4 Content Filter-Pattern

Das Content Filter-Pattern übernimmt genau die gegenteilige Funktion des Content Enrichers: bestimmte Bestandteile einer Nachricht müssen entfernt bzw. die Struktur der Nachricht vereinfacht werden (Hohpe & Woolf 2004, S. 342). Entsprechend einfach lässt sich das Pattern nach BPMN überführen. Die neue Aufgabe ist dabei wie in Abbildung 93 gezeigt zu verwenden.

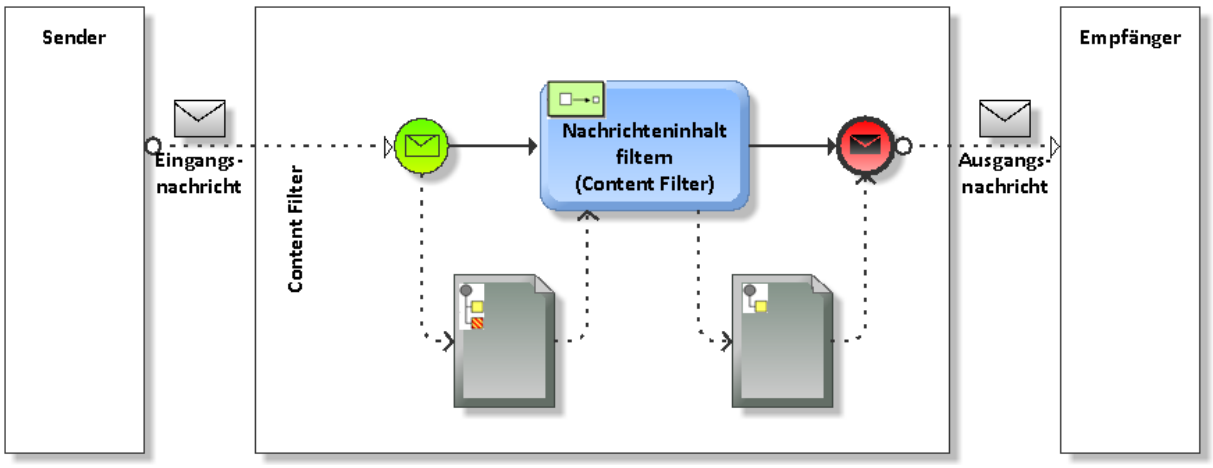



*Abbildung 93: Content Filter-Pattern*

Zur Laufzeit muss lediglich bekannt sein, wie die Filterfunktion implementiert ist. Da es sich hierbei um eine typische Nachrichtentransformation handelt, kann die Implementierung in Form einer Scriptsprache, Java oder in einer XSLT-Transformation erfolgen. Der Verweis auf die Implementierung wird wiederum in den Eigenschaften der Content Filter-Aufgabe hinterlegt. Zur Laufzeit wird die aktuell bearbeitete Nachricht über das Eingangsdatenobjekt an den Content Filter weitergereicht. Dieser bedient sich nun der Filterfunktion, die entsprechend eine Vereinfachung an der Nachricht durchführt. Das dadurch entstandene Resultat wird anschließend an das Ausgangsdatenobjekt weiterreicht und schlussendlich an den Empfänger über das nachrichtenbasierte Endereignis verschickt.

#### 5.6.3.5 Message Translator-Pattern

Das Content Enricher- und das Content Filter-Pattern gehören beide zur Kategorie der Message Transformation-Pattern, die im Wesentlichen die Vermittlung zwischen unterschiedlichen Datenformaten vornehmen. Dazu gehört, als eine Art Verallgemeinerung dieser Pattern-Kategorie, das Message Translator-Pattern (Hohpe & Woolf 2004, S. 86). Es kann immer dann eingesetzt werden, wenn prinzipiell zwischen verschiedenen Formaten transformiert werden muss. Abbildung 94 zeigt das Pattern im Einsatz.

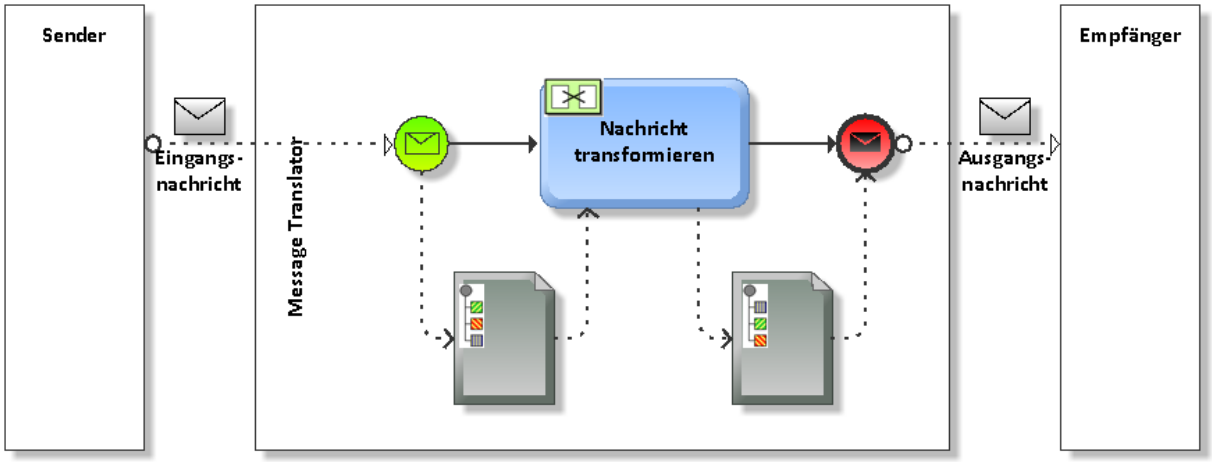



*Abbildung 94: Message Translator-Pattern*

Aufgrund der Fülle möglicher Nachrichtentransformationen macht es keinen Sinn, für jede Einzelheit ein eigenes Symbol einzuführen. Ob dabei Nachrichten-Headerdaten oder die Payload selbst betroffen ist, ob zwischen Objekten und XML vermittelt wird (marshall, unmarshall), ob Nachrichten normalisiert oder in andere Strukturen eingepackt werden, all dies lässt sich im Message Translator über eine Referenz auf die Implementierung der Transformationslogik realisieren. Zur Umsetzung eignen sich einmal mehr Scriptsprachen, Java und XSLT-Transformationen. Die Verarbeitung zur Laufzeit entspricht der des Content Filter-Patterns.

#### 5.6.3.6 Content Based Router-Pattern

Zur Aufgabe eines Content Based Routers zählt der Transfer einer Nachricht zu exakt einem Empfänger abhängig vom Inhalt der eingegangenen Nachricht (Hohpe & Woolf 2004, S. 232). Zur Umsetzung dieser Funktionalität bietet sich die Kombination einer neuen BPMN-Aufgabe vom Typ *Content Based Router* im Zusammenspiel mit einem exklusiven Gateway an, wobei für jedes Gate des Gateways ein Ausdruck

definiert sein muss, der eindeutig bestimmt, ob die Nachricht an den Empfänger weitergeleitet werden soll. Grafisch ist die Funktionsweise in Abbildung 95 dargestellt, in der auch die Verwendung des Standardflusses gezeigt wird, der stets dann zu durchlaufen ist, wenn keine der Bedingungen erfüllt werden konnte:

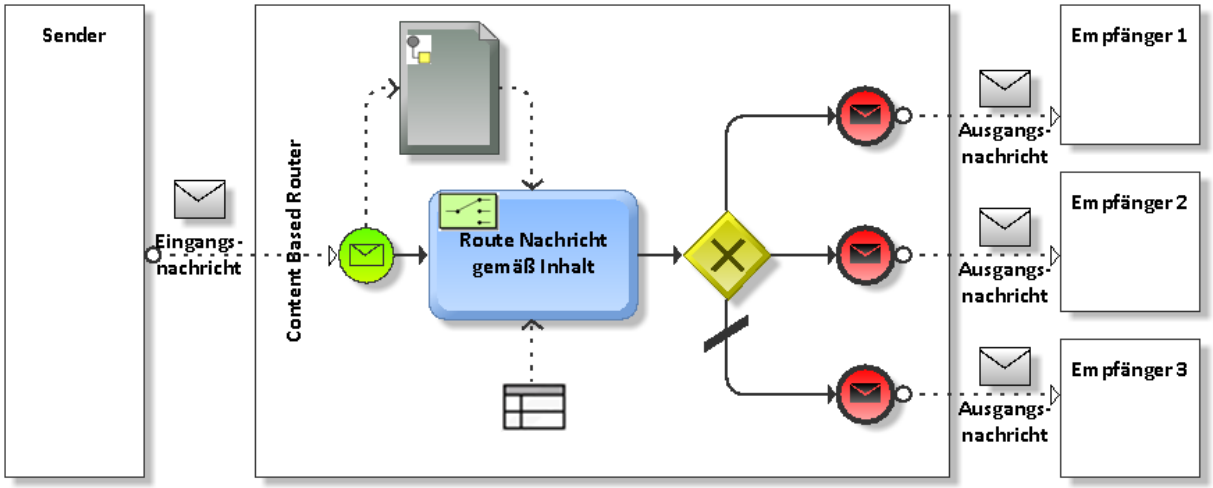



*Abbildung 95: Content Based Router-Pattern*

Für die Formulierung der Bedingungen, die natürlich aufgrund der Forderung, dass nur ein System die Nachricht erhalten soll, konfliktfrei sein müssen, stehen nun wieder verschiedene Möglichkeiten zur Verfügung: ob Scriptsprachen, Java Beans/Java-Programme, XPath-Ausdrücke oder Geschäftsregeln – in den Eigenschaften der Aufgabe *Route Nachricht gemäß Inhalt* wird jeweils die Referenz auf die Implementierung hinterlegt. Grafisch könnte dies durch eine Assoziation veranschaulicht werden, so wie dies in Abbildung 95 für den Einsatz einer Geschäftsregel dargestellt ist.

Ein ähnliches Pattern, das Dynamic Router-Pattern (Hohpe & Woolf 2004, S. 244), kann in vergleichbarer Weise dargestellt werden. Der Unterschied liegt einzig darin, dass die Informationen über die Bedingungen in einer Datenbank abgelegt werden, die beim Systemstart von den beteiligten Zielsystemen gefüllt wird. Die Zielsysteme senden dazu eine Art Anwesenheitsmeldung an den dynamischen Router. Diese Meldung gibt nicht nur Auskunft über die Verfügbarkeit des Empfängers, sondern sie liefert gleichzeitig die Bedingungen mit, unter denen der Empfänger bereit ist, bestimmte Nachrichten anzunehmen. Diese Bedingungen werden vom Router in der Datenbanktabelle eingepflegt und zur Laufzeit ausgewertet. Hinsichtlich der Darstellung als BPMN-Aufgabe ändert sich lediglich die Assoziation: statt der Geschäftsregel, wie in Abbildung 95 gezeigt, ist das in BPMN 2.0 neu eingeführte Datenspeicher-Symbol zu

verwenden ( 

). Ansonsten ist die Funktionsweise des Dynamic Routers zur Laufzeit mit der des Content Based Routers identisch: die eingegangene Nachricht wird gemäß hinterlegter Routinglogik analysiert und je nach Wahrheitswert der verschiedenen Bedingungen an genau einen zugeordneten Empfänger weitergeleitet. Kann keine der Bedingungen erfüllt werden, so wird dem Standardfluss gefolgt und die Nachricht entsprechend behandelt.

#### 5.6.3.7 Message Filter-Pattern

Im Gegensatz zum Content Filter-Pattern, bei dem es um das Ausfiltern einzelner Bestandteile einer Nachricht ging, entfernt das Message Filter-Pattern ungewollte Nachricht komplett aus der Verarbeitung (Hohpe & Woolf 2004, S. 238). Die Verwendung des Filters ist in Abbildung 96 zu sehen:

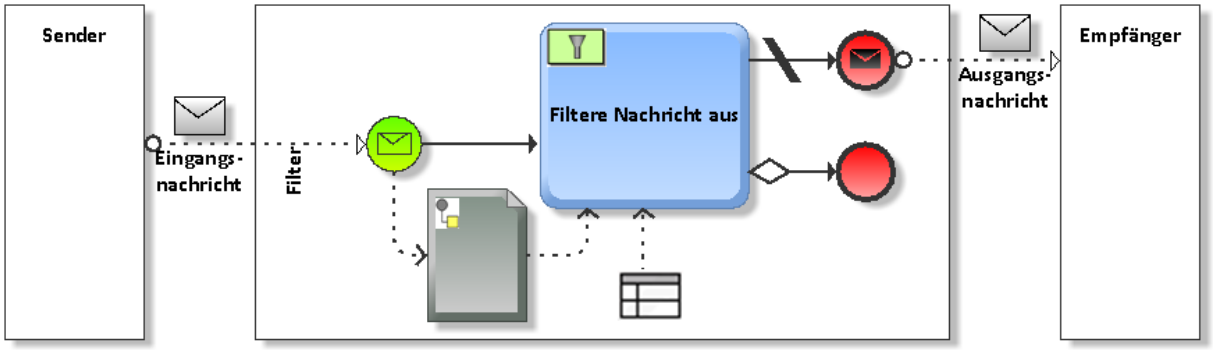



*Abbildung 96: Message Filter-Pattern*

Wie beim Content Based Router-Pattern sind zur Laufzeit lediglich die Bedingungen von Bedeutung, nach denen Nachrichten ausgefiltert werden sollen. Diese können in Form von Scripten, Java Beans, Geschäftsregeln oder, da im Wesentliche auf Nachrichteninhalte zurückgegriffen wird, XPath-Ausdrücken vorliegen, die in den Eigenschaften der Filter-Aufgabe referenziert werden. In Abbildung 96 ist einmal mehr die Verwendung eines Regelwerks beispielhaft dargestellt. Trifft das Filterkriterium für die aktuell in Bearbeitung befindliche Nachricht zu, wird dem bedingten Sequenzfluss gefolgt und das untypisierte Endereignis wird erreicht, mit dem Ergebnis, dass die Nachricht nicht weitergeleitet wird. Andernfalls wird dem Standardsequenzfluss gefolgt und über das nachrichtenbasierte Endereignis erfolgt der Versand der Nachricht an den Empfänger.

#### 5.6.3.8 Recipient List-Pattern

Im Vergleich zum Content Based Router-Pattern übermittelt das Recipient List-Pattern eine Nachricht abhängig von dessen Inhalt an eine Liste von Empfängern (Hohpe & Woolf 2004, S. 250). Zur Realisierung wird eine Kombination der neuen BPMN-Aufgabe vom Typ *Recipient List* und bedingten Sequenzflüssen verwendet, wobei für jeden bedingten Sequenzfluss ein Ausdruck definiert sein muss, der bestimmt, ob die Nachricht an den Empfänger weitergeleitet werden soll, der dem bedingten Sequenzfluss zugeordnet ist. Grafisch ist die Funktionsweise in Abbildung 97 dargestellt. Auch hier dient der Standardfluss dazu, im Falle nicht erfüllter Bedingungen eine Weiterverarbeitung sicherzustellen:

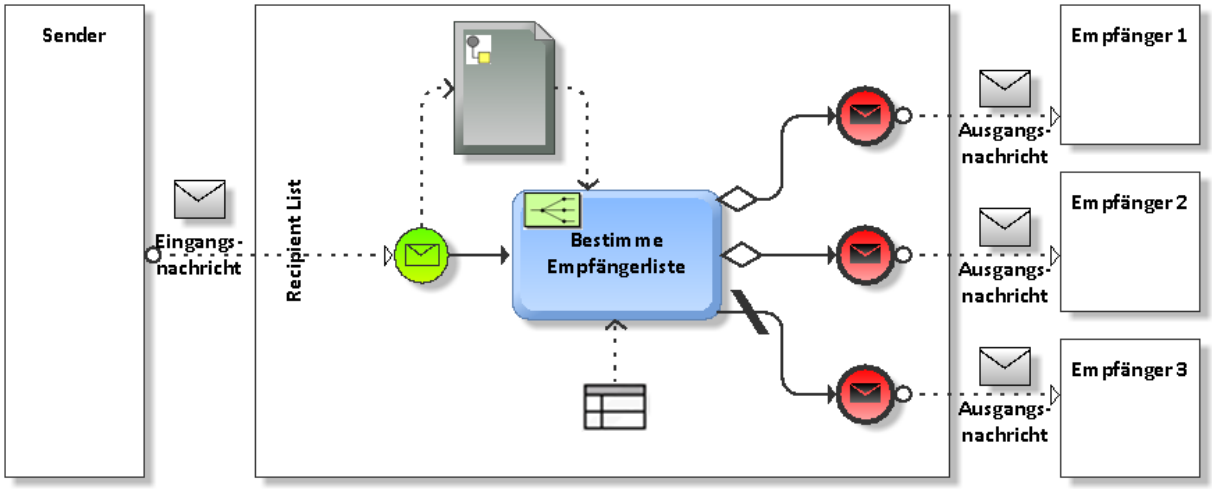



***Abbildung 97: Recipient List-Pattern unter Verwendung von bedingten Sequenzflüssen***

Erneut kann die Bedingung auf vielfältige Weise formuliert werden: Scriptsprachen, Java Beans/Java-Programme, XPath-Ausdrücke oder Geschäftsregeln sind gleichermaßen geeignet. In den Eigenschaften der Aufgabe *Bestimme Empfängerliste* wird dazu die Referenz auf die Implementierung hinterlegt. In Abbildung 97 ist beispielsweise der Einsatz einer Geschäftsregel dargestellt. Zur Laufzeit wird die Routinglogik berechnet und je nach Bedingung an den oder die zugeordneten Empfänger weitergeleitet. Dem Standardfluss wird immer dann gefolgt, wenn zuvor kein Empfänger ermittelt werden konnte.

Alternativ kann das Recipient List-Pattern auch unter Einsatz eines inklusiven Gateways umgesetzt werden. Diese Möglichkeit ist in Abbildung 98 zu sehen.

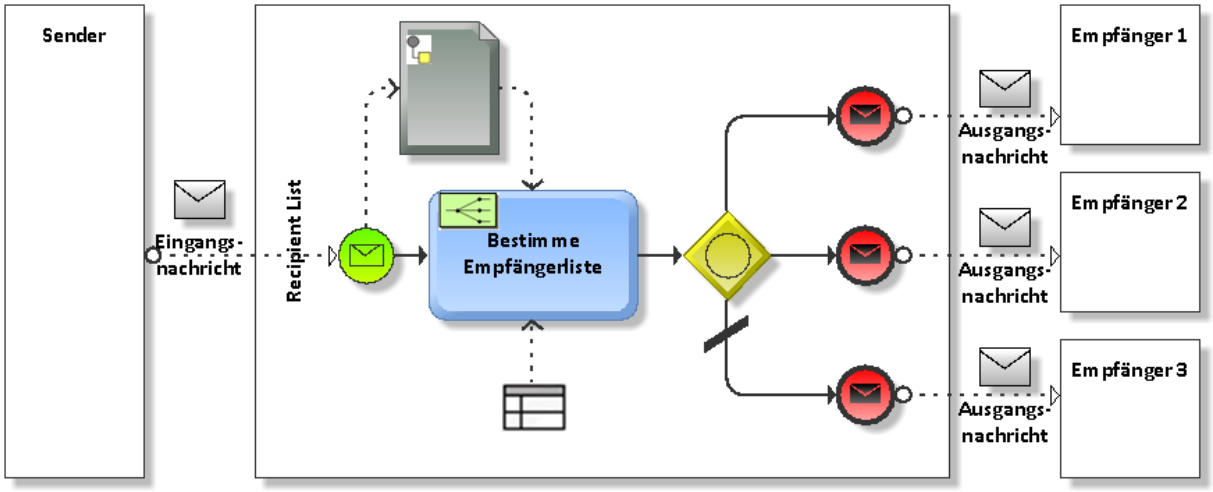



*Abbildung 98: Recipient List-Pattern unter Verwendung des inklusiven Gateways*

#### 5.6.3.9 Resequencer-Pattern

Das Resequencer-Pattern gehört zu den sogenannten zustandsbehafteten Pattern, wie dies auch schon beim Aggregator-Pattern erörtert wurde: eine Sequenz von Nachrichten soll den Empfänger in einer wohldefinierten Reihenfolge erreichen, es ist aber nicht sichergestellt, dass die Nachrichten bereits geordnet eintreffen. Das Resequencer-Pattern hat folglich die Aufgabe zu erfüllen, die einzelnen Nachrichten in die korrekte Reihenfolge zu bringen (Hohpe & Woolf 2004, S. 284). Von daher muss das Pattern solche Nachrichten zwischenspeichern können, die außer der Reihe eingetroffen sind. Sind die noch fehlenden Nachrichten ebenfalls eingetroffen, kann die gesamte zusammengehörige Sequenz verschickt werden. Der Resequencer puffert also nicht alle Nachrichten, die eine Sequenz ausmachen, sondern verschickt diese sofort, sofern es die Sequenz, die durch Sequenznummern innerhalb der Nachrichten festgelegt wird, erlaubt. Es macht also Sinn, das Pattern wieder in zwei Teile aufzuteilen: das Modell aus Abbildung 99 zeigt die Verwendung des Patterns in Form einer neuen Resequencer-Aufgabe, während Abbildung 100 die detaillierte Implementierung des Patterns veranschaulicht.

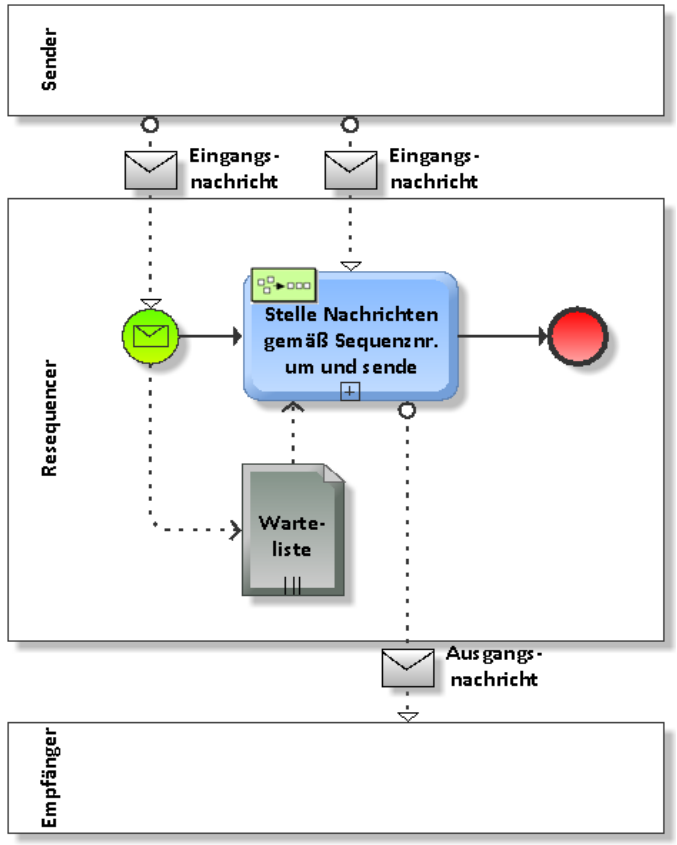



*Abbildung 99: Resequencer-Pattern in kompakter Darstellung*

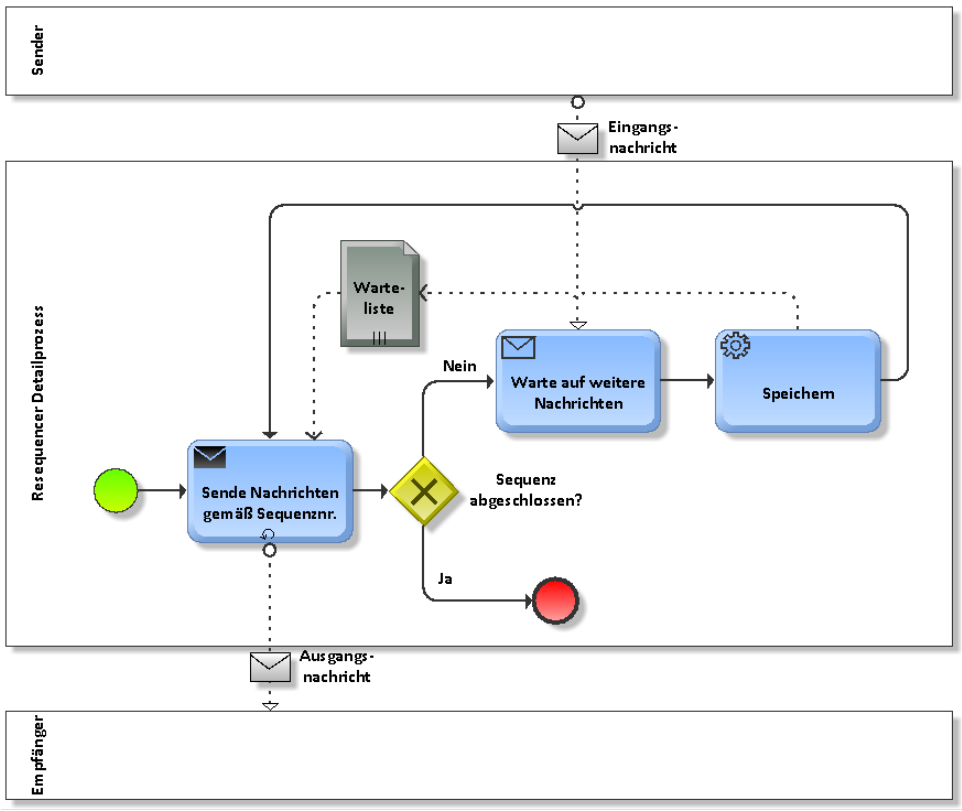



*Abbildung 100: Details des Resequencer –Teilprozesses*

Die Funktionsweise des Resequencer-Patterns zur Laufzeit lässt sich nun wie folgt erklären: die Eingangsnachricht des Senders aus Abbildung 99 startet die Verarbeitung. Die Nachricht wird in ein Datenobjekt namens *Warteliste* umkopiert. Die Warteliste sammelt folglich sämtliche Nachrichten, die noch nicht an den Empfänger verschickt wurden. Außerdem stellt die Liste die Verbindung zur Teilprozessimplementierung aus Abbildung 100 her, die auf dasselbe Datenobjekt zurückgreift. Die Sendeaufgabe aus Abbildung 100 leitet nun so viele Nachrichten weiter, wie Nachrichten mit aufeinanderfolgenden Sequenznummern vorhanden sind. Verschickte Nachrichten werden unmittelbar aus der Liste entfernt. Dabei kann es durchaus passieren, dass keine Nachricht verschickt wird, da die Sequenz aufgrund ausstehender Nachrichten unterbrochen ist. Ist der Sendevorgang abgeschlossen (entweder aufgrund einer leeren Liste oder einer Lücke in den Sequenznummern), so wird am exklusiven Gateway überprüft, ob der Resequenzer beendet werden kann. Dies ist immer dann der Fall, wenn die Warteliste leer und sämtliche erwartete Nachrichten auch eingetroffen sind. Die Anzahl erwarteter Nachrichten wird bei Nachrichtensequenzen üblicherweise als Bestandteil des Nachrichtenheaders mitgeliefert. Jeder Nachricht verfügt also über die Information, aus wie vielen Teilen die Sequenz in Summe besteht. Ist die Sequenz noch nicht abgeschlossen, so muss auf weitere Nachrichten gewartet werden. Verantwortlich hierfür ist die Empfangsaufgabe *Warte auf weitere Nachrichten*. Wird sie verlassen, ist eine neue Nachricht eingetroffen, die wiederum in die Warteliste durch die Service-Aufgabe *Speichern* aufgenommen wird. Der Zyklus kann nun von vorn beginnen.

Zur Konfiguration des Patterns ist es lediglich notwendig, die Felder innerhalb der Nachricht anzugeben, die der Sequenznummer und der Gesamtzahl der Nachrichten, die eine Sequenz ergeben, entsprechen.

Ein Problem gilt es allerdings noch zu lösen. Das Problem ist bereits während der Diskussion des Aggregator-Patterns aufgetaucht und hängt mit der Position des Resequencer-Patterns innerhalb von Verarbeitungsketten zusammen: wird der Resequencer in einer Verarbeitungskette mit anderen Enterprise Integration Pattern verwendet und der Resequencer steht dabei **nicht** am Anfang dieser Kette, wie dies in Abbildung 99 angenommen wurde, so kann in der Implementierung nicht davon ausgegangen wer-

den, dass die erste Nachricht bereits in der Warteliste vorliegt. Stattdessen wird ein vorgeschaltetes Pattern eine Liste von Nachrichten erzeugen, die anschließend vom Resequencer explizit gelesen und sortiert werden muss. Von daher ist für diese Fälle die Implementierung mit dem Empfang der Nachricht zu beginnen. Diesem Sachverhalt wird in der Implementierung aus Abbildung 101 Rechnung getragen.

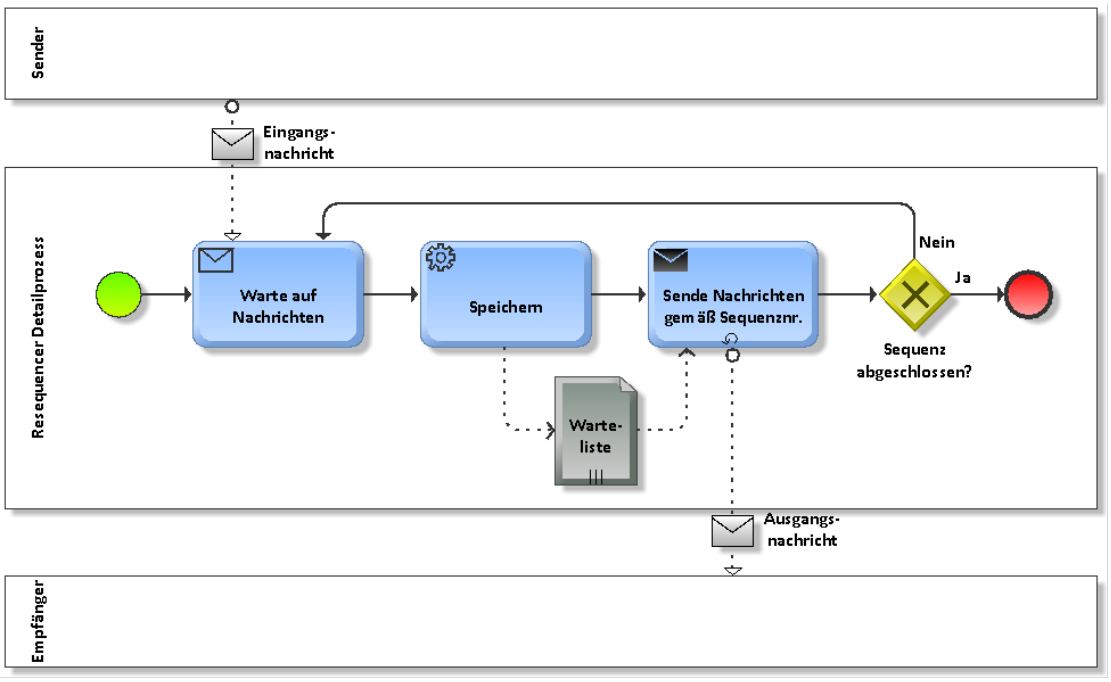



*Abbildung 101: Resequencer-Implementierung mit Nachrichtenempfang als erstem Schritt*

Die grundlegende Logik ändert sich durch diese Umstellung nicht. Es wird lediglich sichergestellt, dass der Resequencer mit dem Nachrichtenempfang beginnt und erst danach die Versandschleife folgt.

#### 5.6.3.10 Splitter-Pattern

Einzelne Nachrichten können in realen Szenarien oft sehr komplex werden. Auftragsdaten beispielsweise bestehen in der Regel aus einem Auftragskopf und mehreren Auftragspositionen. Sowohl der Auftragskopf als auch die Positionen setzen sich ihrerseits wieder aus einer Vielzahl weiterer Segmente zusammen. Wenn eine solche Nachricht verarbeitet werden soll, empfiehlt es sich oft, die Nachricht in ihre Bestandteile zu zerlegen, damit die Einzelteile individuell, gegebenenfalls sogar parallel zur Performancesteigerung, bearbeitet werden können. Genau diese Aufgabe übernimmt

das Splitter-Pattern (Hohpe & Woolf 2004, S. 259). Abbildung 102 veranschaulicht die Verwendung des Patterns:

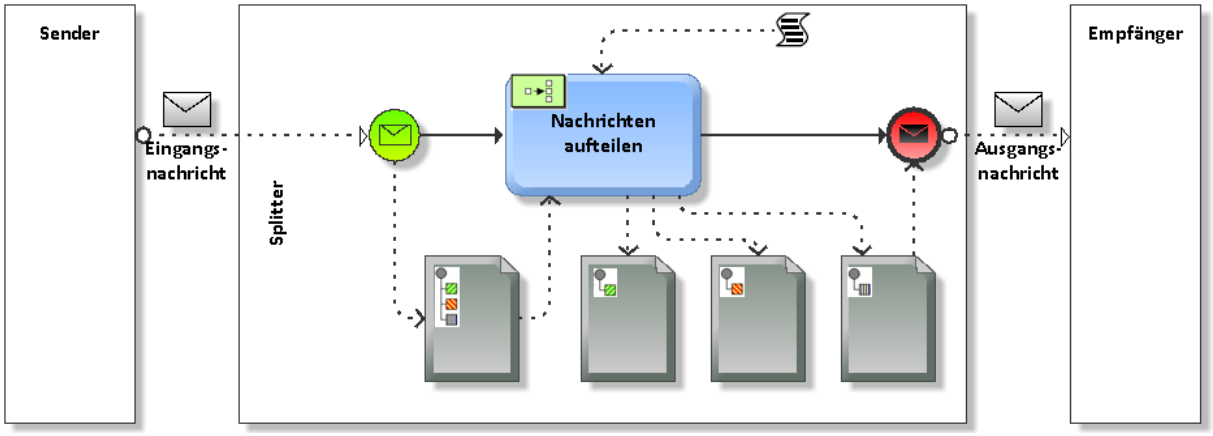



*Abbildung 102: Splitter-Pattern*

Eine eintreffende Nachricht wird über ein Datenobjekt dem Splitter zur Verfügung gestellt. Dieser teilt anschließend die eingegangene komplexe Struktur in verschiedene Fragmente auf. Jedes dieser Teile kann nun im weiteren Prozessverlauf einzeln verarbeitet werden. In der dargestellten Abbildung 102 wird beispielhaft lediglich ein Nachrichtenfragment an den Empfänger weitergereicht.

Die einzige Information, die der Splitter zur korrekten Aufteilung der empfangenen Struktur benötigt, ist die Art und Weise, wie die Nachricht zu zerlegen ist. Dies kann auf vielfältige Weise geschehen und ist abhängig vom konkreten Aufbau der Nachricht. So kann jede Zeile einer Textnachricht genau einen Datensatz repräsentieren und der Splitter erkennt an der Zeilenendemarkierung den Abschluss eines Eintrags. Oder die einzelnen Bestandteile sind durch Sonderzeichen markiert (@, ;), die dem Splitter bekanntgemacht werden müssen. Bei XML-basierten Nachrichten kann ein bestimmtes Tag für die Aufteilung verantwortlich sein. Folglich sind diese Informationen dem Splitter bei dessen Konfiguration mitzugeben. In besonders komplexen Fällen sind auch selbst entwickelte Java Beans denkbar, die für die Zerlegung verantwortlich sind. Wie auch immer die konkrete Umsetzung erfolgt, im Modell lässt sich dies durch eine Assoziation, beispielsweise zu einem Script wie in Abbildung 102 gezeigt, realisieren. Dieser Assoziation müssen dann die Eigenschaften zugewiesen werden, die die Art der Zerlegung repräsentieren (z.B. Referenz auf ein Java Bean oder das Sonderzeichen oder das Tag usw.). Derart konfiguriert kann der Splitter dann seine Aufgabe erfüllen.

#### 5.6.3.11 Composed Message Processor-Pattern

Die besondere Leistungsfähigkeit der bisher erarbeiteten Pattern kommt dann zur Entfaltung, wenn sie kombiniert eingesetzt werden. Hohpe & Woolf schlagen als Beispiel für ein solches Pattern das Composed Message Processor-Pattern vor, bei dem das Splitter-, Content Based Router- und Aggregator-Pattern in einer Sequenz eingesetzt werden (Hohpe & Woolf 2004, S. 294ff). Abbildung 103 zeigt die Abfolge der genannten Pattern unter Einsatz der von Hohpe und Woolf vorgeschlagenen Symbol-Notation im Original.

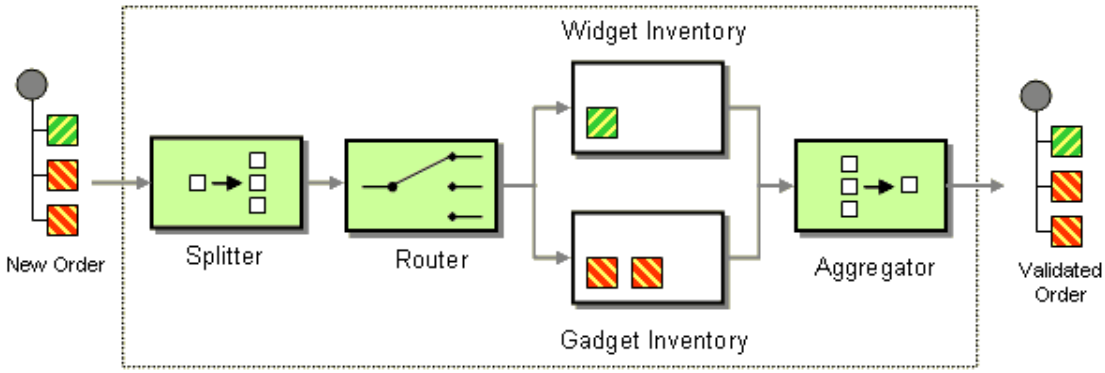



***Abbildung 103: Composed Message Processor-Pattern (aus Hohpe & Woolf 2004, S. 296)***

Wie Abbildung 103 zu entnehmen ist, durchläuft eine eingehende Nachricht einen Splitter, der zunächst die Nachricht in einzelne Bestandteile zerlegt. Anschließend wird jedes Bestandteil (was dem Modell aus Abbildung 103 so auch nicht im Detail zu entnehmen ist) über den Router an zwei mögliche Empfänger verschickt. Deren Antworten sammelt der Aggregator ein und setzt sie zu einer neuen Ausgangsnachricht zusammen. Ein typisches Szenario für die Verwendung dieses Patterns stellt die Validierung der einzelnen Positionen eines Auftrags dar, ob das in der Position angesprochene Produkt also beispielsweise im Lager vorrätig und somit lieferbar ist. In Abbildung 103 werden zwei Arten von Produkten unterschieden, die gegen unterschiedliche Bestände abgeglichen werden müssen. Die um die Validierung ergänzten Daten werden dann als eine Nachricht weitergeleitet.

Wird dieses Modell nun mit der erweiterten BPMN umgesetzt, so ergibt sich das in Abbildung 104 dargestellte Bild.

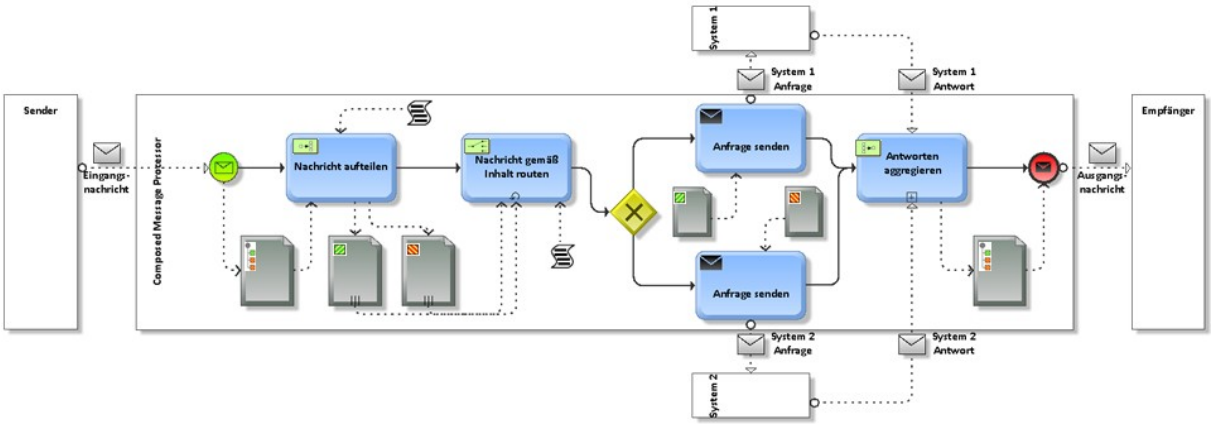



***Abbildung 104: Composed Message Processor-Pattern unter Verwendung der erweiterten BPMN***

Das BPMN-Modell aus Abbildung 104 ist nun im Vergleich zum Modell aus Abbildung 103 wesentlich präziser: es ist zu erkennen, wie die eingehende Nachricht konkret in zwei verschiedene Nachrichtenbestandteile aufgeteilt wird, wobei jeder Bestandteil für sich wieder mehrfach auftreten kann (Listenmarkierung in den dazugehörigen Datenobjekten). Die einzelnen Fragmente werden der Content Based Router-Aufgabe *Nachricht gemäß Inhalt routen* zugeführt, die mit einer Schleifenmarkierung versehen ist. Dadurch wird explizit die wiederholte Ausführung je Nachrichtenfragment unmittelbar ersichtlich. Pro Nachrichtensegment wird an dieser Stelle ein neues Token produziert und an des exklusive Gateway weitergereicht. Das Gateway hat lediglich die Aufgabe, das Ergebnis des Content Based Router's auszuwerten und dem entsprechend das jeweilige Datenfragment an die dazugehörige Sendeaufgabe weiterzuleiten. Im Modell sind

- die Sendevorgänge,
- die einzelnen Nachrichtenbestandteile, die von der jeweiligen Sendeaufgabe verschickt werden,
- sowie die dadurch involvierten Systeme

ebenso klar erkennbar wie die Antwortnachrichten und deren Zusammenführung in der Aggregator-Aufgabe *Antworten aggregieren*. Der Aggregator wird schließlich dann beendet, wenn er exakt so viele Nachrichten empfangen hat wie Anfragen ursprünglich an die beteiligten Systeme verschickt wurden (Endkriterium ist also das Erreichen einer bestimmten Gesamtnachrichtenanzahl). Werden anschließend die Pattern, Schnittstellen, Kommunikationsendpunkte, Gateways usw. wie bei der Diskussi-

on der einzelnen Pattern beschrieben konfiguriert, so ist dieses Modell danach unmittelbar ausführbar.

#### 5.6.3.12 Scatter-Gather-Pattern

Das Scatter-Gather-Pattern ist ein weiteres Beispiel für eine zusammengesetzte Verarbeitungssequenz. Es kombiniert den parallelen Versand einer Nachricht an mehrere Systeme (Broadcast) mit einem Aggregator, der die Antworten der beteiligten Systeme einsammelt und zu einer Ausgangsnachricht zusammenfügt (Hohpe & Woolf 2004, S. 298). Das dazugehörige Originalmodell basierend auf der Pattern-Notation ist in Abbildung 105 dargestellt.

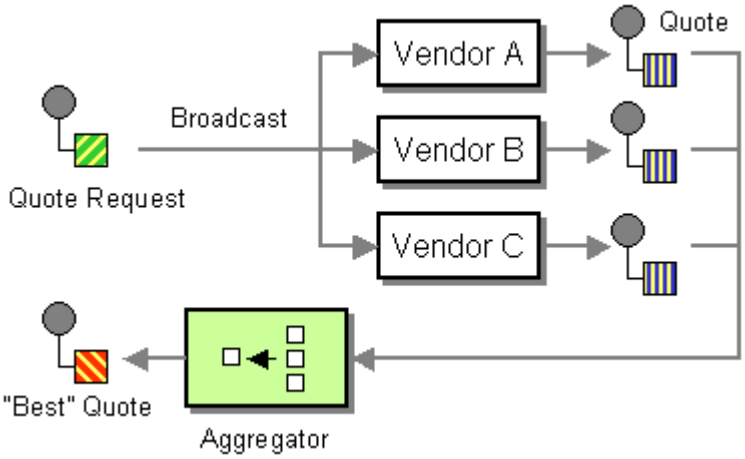



***Abbildung 105: Scatter-Gather-Pattern (aus Hohpe & Woolf 2004, S. 298)***

Abbildung 105 verwendet als Beispiel zur Verdeutlichung des Scatter-Gather-Patterns das Einholen von Angeboten. Die Angebotsanfrage wird per Broadcast an drei Händler verschickt. Die eintreffenden Antworten wiederum werden durch den Aggregator gebündelt und letztendlich als eine Nachricht mit dem besten Angebot an den ursprünglichen Aufrufer zurückgesendet.

Für die Umsetzung dieses Sequenzflusses mit der erweiterten BPMN wird kein separates Pattern für den Broadcast-Teil von Scatter-Gather benötigt, da diese Funktionalität semantisch identisch durch das parallele Gateway abgedeckt wird (siehe Abbildung 106). Das Einsammeln der Antworten übernimmt erneut der Aggregator-Teilprozess, der sich auch in diesem Fall bei Erreichen einer bestimmten Nachrichtenzahl beendet.

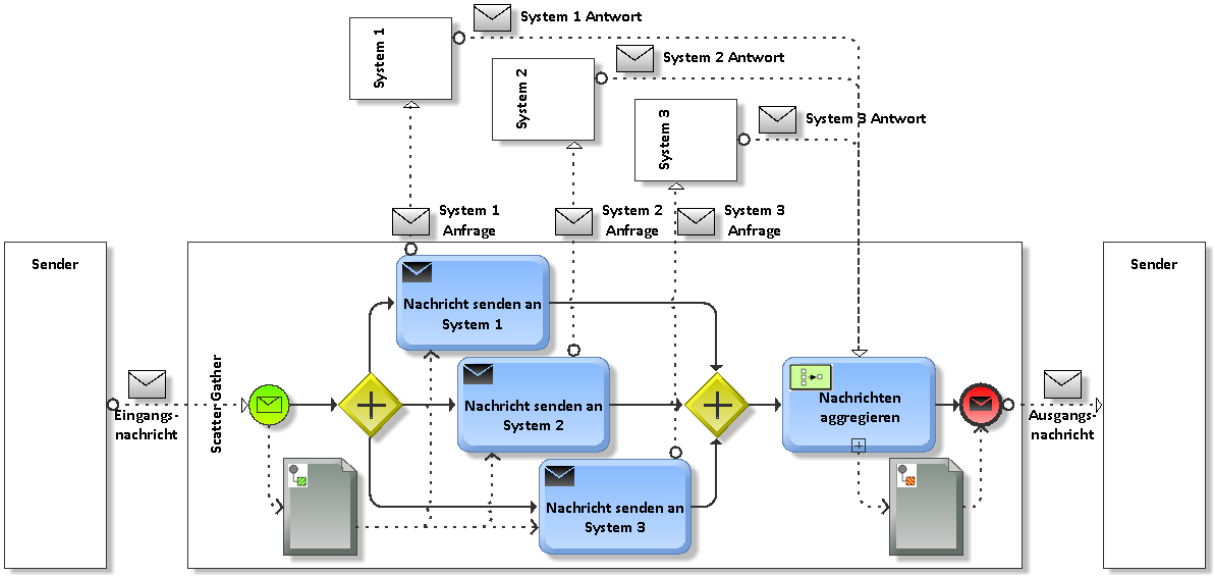



*Abbildung 106: Scatter Gather-Pattern unter Verwendung der erweiterten BPMN*

### 5.6.3.13 Ergänzungen für Nachrichten

Hohpe und Woolf führen in ihrem Buch neben den verschiedenen Integrationspattern auch neue Symbole für verschiedene Nachrichtentypen ein, um diese von ihrem Inhalt her besser unterscheiden zu können. Dabei wird zwischen Kommando-Nachrichten (Hohpe & Woolf 2004, S. 145), Dokument-Nachrichten (S. 147) und Ereignis-Nachrichten (S. 151) unterschieden. Kommando-Nachrichten sind immer dann zu verwenden, wenn in einem Zielsystem eine bestimmte Prozedur/Methode aufzurufen ist. Ein klassisches Beispiel ist das Anlegen eines Auftrags über die Kommando-Nachricht *createOrder*. Als Antwort auf eine Kommando-Nachricht wird in der Regel mit einer Dokument-Nachricht reagiert. Dokument-Nachrichten werden aber nicht nur im Zusammenhang mit Kommando-Nachrichten verwendet. Sie sind auch dann sinnvoll einsetzbar, wenn einfach nur Daten zwischen zwei Systemen ausgetauscht werden müssen (z.B. die Daten eines Kunden oder eines Auftrags). Die Ereignisnachricht kommt immer dann zum Einsatz, wenn Zustandsveränderungen an interessierte Systeme mitzuteilen sind, wobei der Sender die Empfänger nicht kennt. Typische Ereignisse sind Veränderungsmitteilungen an Objekten wie z.B. eine Adressänderung oder aber auch das Löschen einer Auftragsposition.

BPMN kennt zwar das Nachrichtensymbol in Form eines Briefumschlags (✉), es wird aber nicht weiter nach unterschiedlichen Nachrichtentypen unterschieden. Die von Hohpe und Woolf vorgeschlagene Unterscheidung macht aber durchaus Sinn und liefert wichtige Zusatzinformationen, so dass es durchaus angebracht ist, die Nach-

richtentypen auch in der BPMN zu unterscheiden, insbesondere dann, wenn mit ihr Integrationsszenarien umgesetzt werden, an denen unterschiedlichste Nachrichten beteiligt sind. Von daher liegt es nahe, die Symbole für die unterschiedlichen Nachrichtenarten mit dem Nachrichtensymbol der BPMN zu verbinden, so dass die in Abbildung 107 dargestellten neuen Symbole entstehen.


*Abbildung 107: Unterscheidung verschiedener Nachrichtenarten*

#### 5.6.3.14 Verwendung der erweiterten BPMN in konkreten Szenarien

Zum Abschluss der Diskussion möglicher BPMN-Erweiterungen zur prägnanteren Umsetzung systemzentrischer Prozesse sollen zwei Szenarien diskutiert werden, die abermals den konkreten Einsatz der neuen Notation veranschaulichen. Das erste Beispiel stammt aus einem Tutorial zur Open Source-Lösung *Apache Camel*. Apache Camel ist ein Integrationsframework, dass als fundamentalen Bestandteil auf die von Hohpe & Woolf vorgeschlagenen Enterprise Integration Patterns setzt (Camel 2011). Anstey 2009 hat in seiner Einführung zu Camel eine Nachrichtenbearbeitungssequenz gewählt, die in Abbildung 108 dargestellt ist.

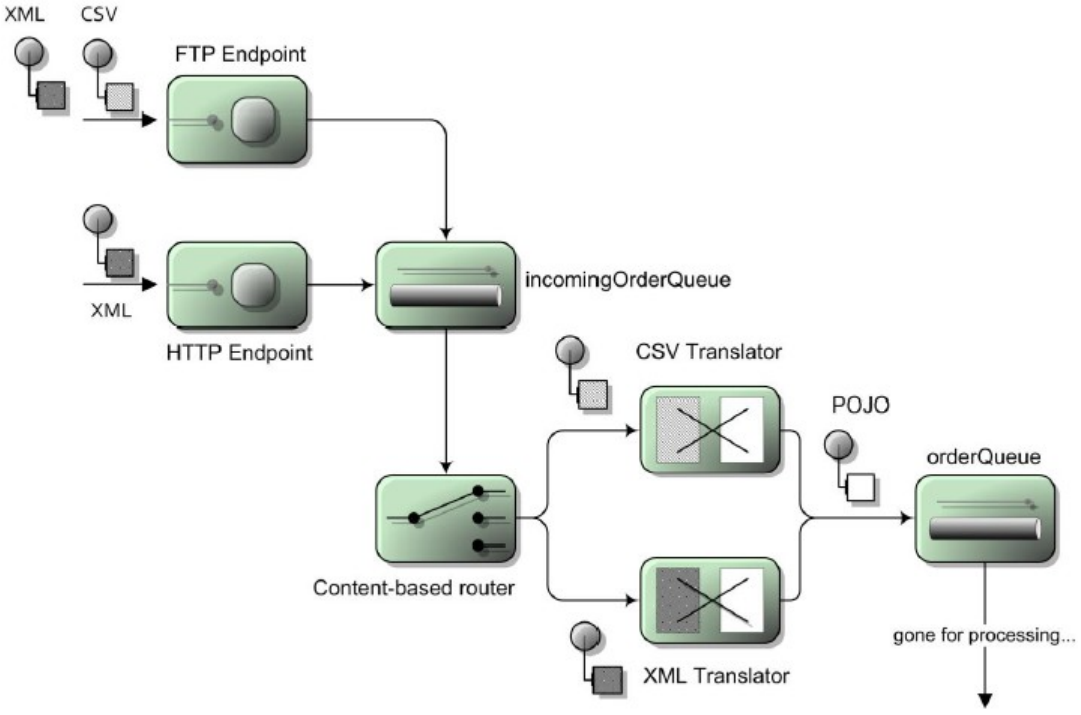



*Abbildung 108: Nachrichtenbearbeitung unter Verwendung der Pattern-Symbole (Anstey 2009)*

Der Ablauf des Szenarios gestaltet sich nun wie folgt (aus Anstey 2009):

Bei der fiktiven Firma, die diesen Prozess implementiert hat, handelt es sich um einen Ersatzteillieferanten für Motorräder. Kunden dieser Firma sind primär Motorradhersteller. Im Laufe der Zeit hat sich die Art und Weise geändert, wie bei dieser Firma Ersatzteile bestellt werden können: ursprünglich wurde eine solche Bestellung als CSV-Datei (comma-separated values) auf einem FTP-Server hinterlegt. Im Zuge der SOA-Bewegung wurde das Angebot möglicher Formate um das XML-Format erweitert, so dass die Firma heute sowohl das CSV-Format als auch das XML-Format für Bestellungen anbietet. Zusätzlich kann das XML-Format neben dem klassischen File-Transfer auch über HTTP übermittelt werden. Neukunden empfiehlt die Firma, nur noch den HTTP-Transfer von XML-Dateien zu verwenden, aber aufgrund vertraglicher Verpflichtungen muss sie sowohl die alten Protokolle als auch das alte CSV-Datenformat unterstützen. Intern werden die Bestelldaten ausschließlich im Format einfacher Java-Objekte, sogenannter POJOs (Plain Old Java Objects), weiterverarbeitet. Von daher muss eine Transformation der eingegangenen Bestellungen nach POJOs sowohl für XML- als auch für CSV-Inhalte vorgesehen werden. Abbildung 109 zeigt nun den dazugehörigen äquivalenten Prozess umgesetzt mit der erweiterten BPMN.

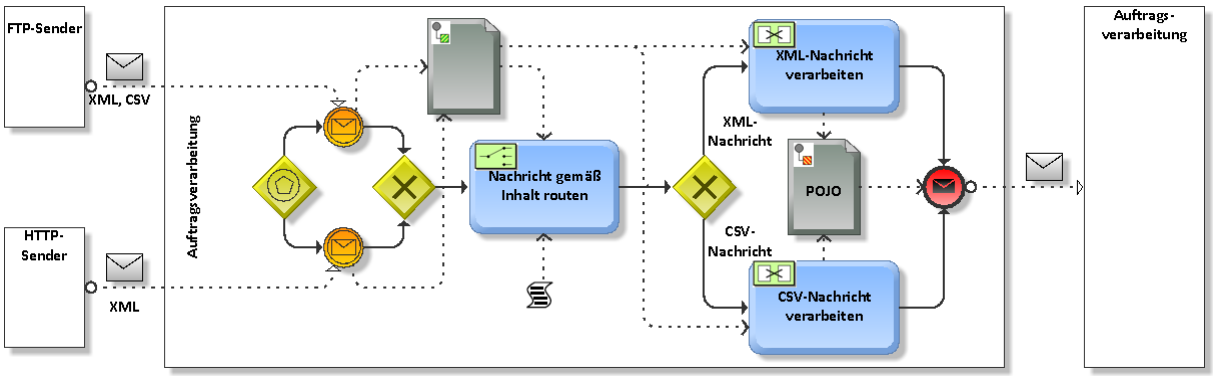



*Abbildung 109: Nachrichtenbearbeitung unter Verwendung der erweiterten BPMN*

Vergleicht man beide Modelle hinsichtlich der Verwendung der verschiedenen Integrationspattern, so lässt sich Folgendes feststellen (vgl. auch Anstey 2009, S. 5) :

1. Die FTP- und HTTP-Endpunkte aus Abbildung 108 können in BPMN durch unterschiedliche Pools dargestellt werden. Als Eigenschaften der Pools ist die Konfiguration technischer Details denkbar, wie z.B. die IP-

Adresse des zuständigen FTP-Servers sowie Informationen zur Anmeldung am FTP-Server (beispielsweise Benutzerkennung/Password, X.509-Zertifikate oder SAML-Tickets).

2. Der Verarbeitungsprozess soll gestartet werden, sobald über einen der möglichen Eingangskanäle eine Nachricht eintrifft. In Abbildung 108 ist dies über eine gemeinsame *incomingOrderQueue* geregelt. Im dazugehörigen BPMN-Modell wird dafür das instanziierende exklusive ereignisbasierte Gateway verwendet. Unabhängig davon, über welchen Weg die Nachricht eintrifft, wird der Prozess gestartet.
3. Die eingegangene Nachricht wird nun über den Content Based Router geleitet, der sie abhängig von ihrem Format an den passenden Message Translator weiterreicht. Im BPMN-Modell ist ein zusätzliches Gateway notwendig, da die Semantik, dass ausschließlich einem Pfad gefolgt wird, nur über dieses Gateway realisiert werden kann. In Abbildung 108 führen die Verbindungen aus dem Content Based Router direkt zu den jeweiligen Translator-Pattern. Dies ist in BPMN so nicht möglich, da damit ein Verhalten wie bei einem parallelen Gateway ausgedrückt würde: beide Pfade erhielten ein Token, was aber in obigen Szenario nicht erwünscht ist.
4. Die Message Translator-Pattern wandeln die jeweiligen Eingangsformate in ein POJO um. Das POJO wird in Abbildung 108 in einer Queue gespeichert, aus der anschließend die Weiterverarbeitung erfolgen kann. Dieser Sachverhalt ist in Abbildung 109 durch Hinzufügen eines weiteren Pools repräsentiert. Details der verwendeten Queue würden abermals als Eigenschaften des Pools hinterlegt.

Auch dieses Beispiel zeigt einmal mehr, wie durch Verwendung der erweiterten BPMN klassische Integrationsszenarien sehr prägnant umgesetzt werden können und dabei gleichzeitig die Kompaktheit der ursprünglichen Pattern-Notation nicht verloren geht.

Bei dem zweiten Beispiel für die Umsetzung von Integrationsszenarien mit der erweiterten BPMN handelt es sich um einen SAP-internen Testprozess für das Produkt SAP NetWeaver Process Integration. Er wird stets für Regressionstests herangezogen,

da er eine Vielzahl unterschiedlichster Nachrichtenverarbeitungsschritte vereint. Das BPMN-Modell ist in Abbildung 110 dargestellt.

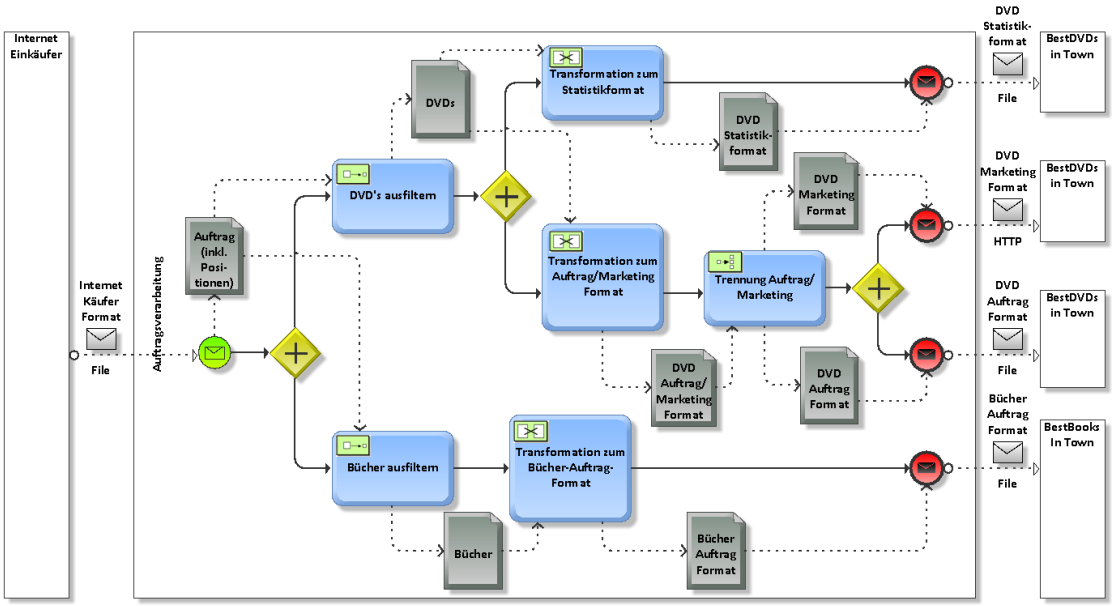



***Abbildung 110: Komplexe Bestellabwicklung unter Verwendung der erweiterten BPMN***

Der in Abbildung 110 dargestellte Prozess adressiert das folgende fachliche Problem: ein Internetshop-Dienstleister für Bücher und DVDs erhält Bestellungen, die er allerdings nicht selbst bedient, sondern diese an zwei Lieferanten weiterreicht, die sich auf Bücher (Lieferant *BestBooks in Town* in Abbildung 110) bzw. DVDs (Lieferant *BestDVDs in Town* in Abbildung 110) spezialisiert haben. Der *BestBooks in Town*-Lieferant will lediglich die Buchbestellungen in seinem Zielformat auf einem von ihm bereitgestellten Fileserver zur Verfügung gestellt bekommen. Lieferant *BestDVDs in Town* ist da schon anspruchsvoller: er will neben den normalen Bestelldaten zusätzlich Marketing-Informationen darüber, welche Produkte von dem Einkäufer geordert wurden. Außerdem benötigt er aggregierte Daten für seine Statistik, die darüber Auskunft gibt, an welchem Tag für welchen Gesamtwert eingekauft wurde. Erschwerend kommt hinzu, dass die Statistik-Daten in einem anderen System als die Auftrags- und Marketingdaten zu hinterlegen sind. Die Auftrags- und Marketingdaten werden zwar an ein und dasselbe ERP-System geschickt, die zu verwendenden Kommunikationsprotokolle sind hingegen unterschiedlich: die Übertragung der Marketingdaten hat per HTTP zu erfolgen, während die Auftragsdaten per Filetransfer zu übermitteln sind.

Zur Lösung dieses Problems werden eingehende Bestellungen nun wie folgt prozessiert: die per Filetransfer eingegangene Bestellung wird zunächst in ein internes Datenobjekt *Auftrag* umkopiert. Es stellt die Arbeitsgrundlage für den Integrationsprozess dar. Am parallelen Gateway erfolgt die Unterscheidung zwischen der Bearbeitung der DVDs und der der Bücher. Ein Content Filter-Pattern (nicht zu verwechseln mit dem Message Filter-Pattern) je Zweig sorgt für eine entsprechende Filterung: für den Bücherzweig werden die Bücher aus der Ursprungsnachricht extrahiert, für den DVD-Zweig entsprechend die DVDs. Die Ergebnisse der Filterung werden in eigenständige Datenobjekte ausgelagert, auf denen der jeweilige Zweig nun autark operieren kann.

Im Bücherzweig muss nach der Extraktion lediglich eine Transformation in das Zielformat des Bücher-Lieferanten erfolgen. Die DVD-Bearbeitung wird hingegen mit einem weiteren parallelen Gateway fortgesetzt, deren Transformationsschritte letztendlich Umformungen zu den erwarteten Zielformaten des Lieferanten vornehmen. Der obere Zweig des parallelen Gateways nimmt sich dabei der Daten für die Statistikdatei an. Ist das Zielformat für die Statistik einmal erstellt, kann die dazugehörige Datei sogleich per Filetransfer übermittelt werden. Der untere Zweig transformiert zunächst in ein Datenobjekt, das noch sowohl Auftrags- als auch Marketingdaten umfasst. Da diese Daten aber separat zu versenden sind, muss ein nachfolgender Splitter noch für deren Trennung sorgen. Der Sendevorgang von sowohl Auftrags- als auch Marketingdaten kann dann wiederum parallel erfolgen. Durch das implizite parallele Join-Verhalten von End-Ereignissen wird der Prozess erst dann beendet, wenn auch wirklich alle Token aus dem Prozess verschwunden sind. Ein weiteres zusammenführendes paralleles Gateway ist daher nicht erforderlich.

Die zur Ausführung notwendige Konfiguration ist offensichtlich: die verschiedenen Nachrichtenformate verbergen sich hinter den jeweiligen Nachrichtensymbolen zwischen den Pools. Die Pools selbst umfassen die technischen Informationen zu den Endpunkten (Fileserver-Informationen, HTTP-Adresse,...). Die Konfiguration der einzelnen Pattern wurde im Laufe dieses Kapitels bereits behandelt. In Summe deckt also das Integrationsszenario aus Abbildung 110 die eingangs genannten Anforderungen ab.

#### 5.6.3.15 Anmerkungen zur erweiterten BPMN

Wie die Beispiele gezeigt haben, ist eine Umsetzung von Integrationsszenarien mit Hilfe der vorgeschlagenen erweiterten BPMN relativ einfach möglich. Durch die Verbindung der BPMN mit den Symbolen der Integrationspattern aus Hohpe & Woolf 2004 entsteht eine kompakte grafische Notation für Systemintegrationen, die über die Möglichkeiten der puren Pattern-Notation hinausgeht: Datenflüsse werden ebenso sichtbar wie die involvierten Systeme. Auf der anderen Seite können sich auch Systemintegratoren in der neuen Notation wiederfinden, da sie die verwendeten typischen Pattern-Symbole als Bestandteil der neuen BPMN-Aufgaben und -Teilprozesse wiedererkennen und somit wissen, was konkret im Prozessmodell in der jeweiligen Task technisch umgesetzt wird. Darüber hinaus profitiert die neue Notation von den Vorteilen, die BPMN standardmäßig auszeichnet: unterbrechende als auch nicht unterbrechende Ereignisse können auch den neuen Aufgaben angeheftet werden. Dadurch besteht die Möglichkeit, sehr dediziert auf Fehlersituationen reagieren zu können. In gleicher Weise sind die Markierungen auf die neuen Aufgaben anwendbar: ob Standardschleife, parallele oder sequenzielle Mehrfachausführung oder die Kompensation – im Zusammenspiel mit den Pattern ergeben sich sehr ausdrucksstarke Möglichkeiten, für die die reine Pattern-Notation keine adäquaten Konstrukte vorsieht. Schlussendlich können ganze Pattern-Sequenzen in Teilprozessen zusammengefasst, wiederverwendet und über Ereignisse überwachbar gemacht werden. In Summe ergeben sich dadurch völlig neue Möglichkeiten für Systemintegratoren zur Lösung typischer Integrationsprobleme in Unternehmen mit heterogenen Systemlandschaften und/oder intensiven B2B-Beziehungen mit Kunden, Partnern und Lieferanten.

## 5.7 Flexibilitätsgewinn durch die Verwendung von Regelwerken in Kombination mit analytischen Anwendungen

Regelwerke und deren Ausführbarkeit durch Regelmaschinen sind aus heutigen Lösungen nicht mehr wegzudenken. Sie finden ihren Einsatz immer dort, wo schnelle Anpassungen an sich ändernde Geschäftsbedingungen notwendig werden, also genau in dem Umfeld, in dem sich auch Verbundanwendungen bewegen. Von daher liegt der Einsatz von Regelwerken insbesondere im Composite Applications-Bereich nahe. Dabei kann die sich häufig ändernde Geschäftslogik in Form von Geschäftsregeln formuliert und je nach wirtschaftlicher Entwicklung nachträglich zur Laufzeit angepasst

werden. Es brauchen also keine aufwändigen Implementierungsprojekte gestartet zu werden, um die Strategie des Unternehmens wieder mit den ausgeführten Prozessen in Einklang zu bringen.

Es gibt zahllose Beispiele, die den sinnvollen Einsatz von Regeln nahelegen:

- Validierungsregeln finden bei Formularüberprüfungen ihre Anwendung. Ein weiteres Beispiel ist die Überprüfung, ob Forderungen seitens eines Kunden rechtens sind.
- Berechnungsregeln für Steuern, Preise, Zinsen, Provisionen, Rabatte, Boni, usw.
- Entscheidungsregeln zur Bestimmung, ob jemand Anspruch auf einen Kredit hat oder ob ein Reisender bei der Zollabfertigung genauer untersucht werden soll.
- Empfehlungsregeln, wie ein Patient weiter zu behandeln ist oder beim Online-Einkauf (Kunden, die dieses Produkt erworben haben, haben auch die folgenden Produkte gekauft).
- Personalisierungsregeln, beispielsweise für den Besuch von bestimmten Web-Seiten.
- Internationalisierungsregeln wie Zollvorschriften oder unterschiedliche Behandlung von Versicherungsverträgen.
- Ausnahmeregeln wie saisonale Aktionen, Werbekampagnen oder Spezial-Arrangements.
- Optimierungs- und Konfigurationsregeln wie die Produktkonfiguration.

Letztendlich sind Regeln ein Weg, die Unternehmenspolitik konsistent und für alle nachvollziehbar darzustellen und dabei gleichzeitig deren Umsetzung zu gewährleisten. Allerdings soll in diesem Kapitel nicht auf die grundlegenden Eigenschaften von Regeln eingegangen werden, sondern vielmehr der Nutzen von Regelwerken in Verbundanwendungen, und hier insbesondere im Zusammenspiel mit Prozessen, diskutiert werden. Für einen Einstieg in die Welt der Regeln sei auf Ross 2009 sowie auf das Business Rules Manifest der Business Rules Group (BRG 2003) verwiesen.

Regelwerke und Prozesse bilden eine optimale Symbiose. Ross 2009 betont die Unabhängigkeit der Regeln und damit verbunden die strikte Trennung von Regeln

und Prozessen. Stattdessen sollen Geschäftsregeln den Prozess führen (Ross 2009, BRG 2003) und damit dazu beitragen, dass die eigentlichen Prozesse relativ stabil bleiben können. Was sich hingegen häufig ändern wird, sind die zugrundeliegenden Regeln. Ross verweist dabei wiederum auf Burlton 2001: „If you separate the business rules, you can develop remarkably stable business processes“ Ross 2009, S. 123) und „The really rapid change is in the business rules … not in the business processes.“ (Ross 2009, S. 123)

Doch wie sieht eine konkrete Architektur aus, die von einem solchen Ansatz profitiert? Auch hierzu gibt es in der Literatur bereits interessante Ansätze. Einen guten Überblick einschließlich weiterführender Literatur verschaffen Graml, Bracht und Spies 2008 in ihrem Artikel über den Einsatz von Geschäftsregeln zur Unterstützung agiler Geschäftsprozesse. Ein Geschäftsprozess wird dann als agil bezeichnet, wenn sowohl der Kontroll- als auch der Datenfluss zur Laufzeit geändert werden kann. In besagtem Artikel werden mögliche Lösungen dieses Problems diskutiert. Da für Verbundanwendungen ähnliche Probleme zu lösen sind, werden im Folgenden die im Artikel erläuterten Lösungsvorschläge hinsichtlich ihrer Relevanz für Composite Applications analysiert und die Lösungen, sofern möglich, für Composites passend adaptiert. Graml, Bracht und Spies verwenden zur Modellierung des Geschäftsproblems zwar ebenfalls die BPMN, transformieren diese Modelle zur Ausführung allerdings wieder nach BPEL. Hier erlaubt die Verwendung von BPMN 2.0 neue Alternativen.

Nach der generellen Diskussion über die Verwendung von Geschäftsregeln in Prozessen zur Steigerung der Flexibilität, wird in einem separaten Abschnitt der Einsatz insbesondere in technischen Prozessen betrachtet. Damit erweitert sich das Anwendungsspektrum für Regeln von der Composite-Schicht auch auf die Servicevertrag-Implementierungsschicht. Wo der Einsatz von Regeln im Detail möglich und sinnvoll ist, wird ebenfalls Gegenstand der Betrachtungen sein.

Abschließend wird eine weitere Steigerung des Automatisierungsgrades von Prozessen durch die Kombination von Geschäftsregeln und analytischen Anwendungen angestrebt. Gegenstand dieses Abschnitts ist die Art und Weise der Kopplung von Regeln und analytischen Applikationen, um die Anzahl von interaktiven Schritten in Prozessen weiter zu reduzieren.

### 5.7.1 Einsatz von Geschäftsregeln zur Steigerung der Flexibilität

Dieser Abschnitt orientiert sich an den von Graml, Bracht und Spies (2008) erarbeiteten Lösungen zur Flexibilitätssteigerung von Geschäftsprozessen unter dem Einsatz von Geschäftsregeln. Das Problem bei fest-modellierten Prozessen liegt in den begrenzten Möglichkeiten, laufende Prozesse nachträglich an geänderte Bedingungen anzupassen. Viele der Probleme hängen dabei direkt mit dem Web Service-Modell in SOA-Umgebungen zusammen, da sich der Austausch von Services auf identische Porttypen beschränkt. Erschwerend kommt die starre Definition des Prozessablaufs hinzu, die bereits zur Designzeit festgelegt wird.

Zur Lösung dieser Limitierungen unterscheiden Graml, Bracht und Spies zwei Strategien: Die erste, Flexibilität durch Selektion (*flexibility by selection*), versucht, mögliche zukünftige Prozessänderungen vorherzusehen und diese bereits zur Designzeit im Modell in Form unterschiedlicher Kontrollflusskombinationen zu berücksichtigen. Zur Laufzeit wird dann die zum jeweiligen Kontext passende Variante ausgewählt (selection). Offensichtlich krankt dieser Ansatz an der Unmöglichkeit, sämtliche potenziellen Prozessvarianten vorherzusehen.

Alternative zwei, Flexibilität durch Anpassung (*flexibility by adaption*), basiert auf der Idee, die Änderung von Porttyp, Kontrollfluss und Datenfluss sowohl des Prozessmodells als auch der laufenden Prozessinstanzen direkt während der Laufzeit zu unterstützen. Graml, Bracht und Spies kombinieren in ihrem Lösungsansatz beide Alternativen: erstens in Form von Pattern, um mögliche Anpassung vorherzusehen und zweitens durch die Verwendung von Geschäftsregeln, um die Verwendung u.a. der Pattern zu steuern.

Bei den Regeln werden wiederum die in Schacher und Grässle 2006 genannten drei Kategorien Ableitungs-, Bedingungs- und Prozessregeln unterschieden.

- Ableitungsregeln: repräsentieren Wissen und spezifizieren, wie neue Informationen aus existierenden Informationen hergeleitet werden können. Klassisches Beispiel ist die Preisfindung für bestimmte Kunden.
  Ross 2009 bezeichnet diese Regeln als strukturelle Regeln oder auch als Definitionsregeln, bei deren Ausführung entweder etwas klassifiziert oder berechnet wird.

- Bedingungsregeln: repräsentieren Ausdrücke, die zu jedem Zeitpunkt erfüllt sein müssen. Folglich ist ein typisches Kennzeichen dieser Regeln, dass sie verletzt werden können. So darf beispielsweise für einen bestimmten Kunden der Auftragswert einen vordefinierten Wert nicht überschreiten. Ross 2009 bezeichnet diese Regeln als Verhaltens- oder operative Regeln, die immer dann eine Rolle spielen, sobald Menschen involviert sind.

- Prozessregeln: repräsentieren innerhalb eines Prozesses Logik, die darüber entscheidet, ob gewisse Aktivitäten oder Teilprozesse, abhängig vom aktuellen Prozesskontext, ausgeführt werden oder nicht. So sind insbesondere bei international tätigen Firmen unterschiedliche Prozesse für die verschiedenen Regionen umzusetzen, damit deren gesetzliche Bestimmungen erfüllt werden. Aufgrund des Prozesskontextes kann dann entschieden werden, welches Prozessfragment zur Ausführung gelangt.

Zur Verdeutlichung des Einsatzes der verschiedenen Regelkategorien werden diese im Folgenden auf den bisher in dieser Arbeit verwendeten Bestellprozess angewendet. In Abbildung 111 ist er ohne Details der Servicevertrag-Implementierungsschicht dargestellt.

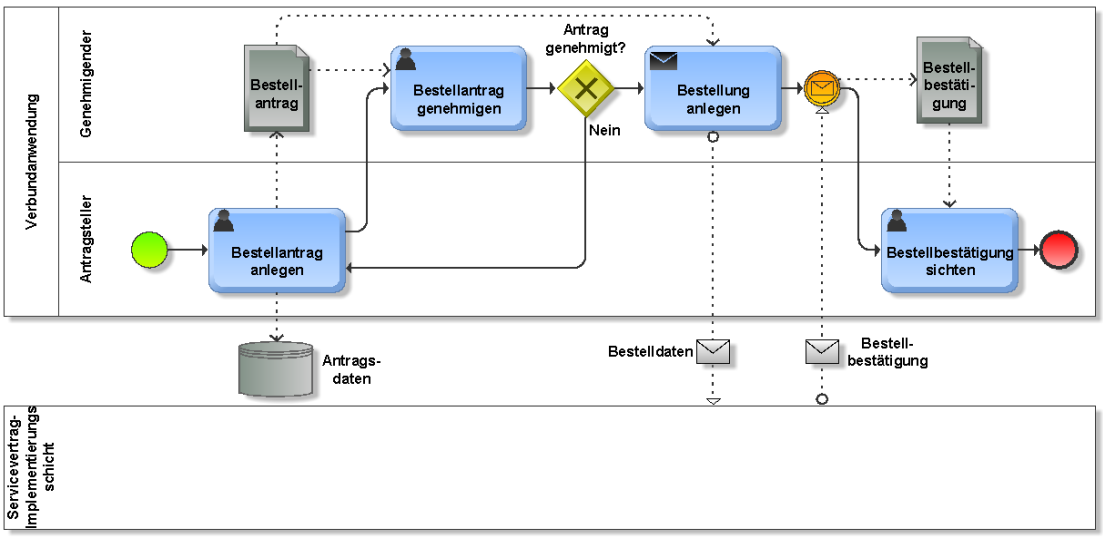



*Abbildung 111: Vereinfachter Bestellprozess ohne Regelverwendung*

Ein wesentliches Ziel jeder Prozessimplementierung ist eine nahezu vollständige Automatisierung der Prozesse ohne Beteiligung des Menschen. Jede menschliche Interaktion im Prozessablauf ist eine potenzielle Fehlerquelle, sorgt für Verzögerungen und erhöht die Prozesskosten. Folglich wird eine weitestgehende Vermeidung von Dialogschritten angestrebt. Der Mensch sollte ausschließlich in Ausnahmesituationen zu Rate gezogen werden. Im Prozess aus Abbildung 111 bietet sich daher die Einführung einer Ableitungsregel unmittelbar vor Aufruf des Genehmigungsschrittes an. Dazu muss lediglich das Wissen des Genehmigenden über die Entscheidungsfindung für diesen konkreten Vorgang in ein Regelwerk überführt werden. Auf diese Weise wird wertvolles Expertenwissen für das Unternehmen formalisiert und wiederverwendbar gemacht. In dem Beispiel könnte die Genehmigung von der Bestellregion und der Auftragsgröße abhängen. Erst ab einem bestimmten Bestellwert soll der Genehmigende hinzugezogen werden. Zur Implementierung reicht in diesem Fall eine einfache Entscheidungstabelle. Abbildung 112 zeigt den neuen Prozessablauf nach Hinzufügen der Regelaufgabe, die erst mit BPMN 2.0 neu hinzugekommen ist.

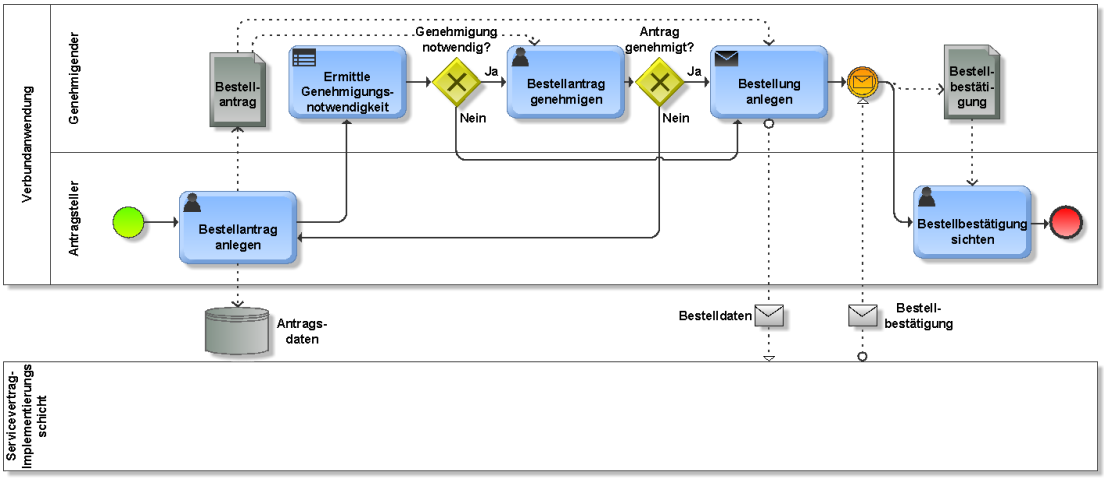



***Abbildung 112: Vereinfachter Bestellprozess mit Regelverwendung***

Nach Eingabe der Bestelldaten ermittelt das Regelwerk, ob eine manuelle Genehmigung notwendig ist. Ist dem nicht so, kann die Bestellung direkt angelegt werden. Dieses Beispiel verdeutlicht zudem das Zusammenspiel zwischen Regelwerk und Prozess: das Regelwerk bereitet die Entscheidung vor, die anschließend vom Prozess ausgewertet wird. Die Regel ist zudem als separater Schritt ausgewiesen und nicht hinter dem Gateway versteckt – ein ebenfalls wichtiges Pattern, damit relevante Informatio-

nen dem Modellbetrachter nicht vorenthalten werden. Milanovic, Gasevic und Wagner 2008, die ebenfalls die Kombination von Prozessen und Regeln untersuchten, empfehlen gar die Erweiterung der BPMN um einen neuen Gateway-Typ, den sie mit Regel-Gateway bezeichnen, und damit eine neue Modellierungssprache namens rBPMN (regelbasierte BPMN) vorschlagen. Doch durch die Kombination von Regelaufgabe und darauffolgendem Gateway wird dieselbe Funktionalität ebenfalls erreicht.

Es wäre auch möglich gewesen, die gesamte Logik der Entscheidungstabelle im Prozessfluss durch kaskadierende Gateways zu modellieren, doch dann wäre während der Prozessausführung keine Einflussnahme mehr möglich gewesen. Durch die Auslagerung der Logik als Geschäftsregeln lassen sich diese nun auch zur Laufzeit an veränderte Rahmenbedingungen anpassen. Auch bereits gestartete Instanzen profitieren von den neu eingebrachten Änderungen. Ein Beispiel für eine einfache Entscheidungstabelle für den Beispielprozess zeigt Abbildung 113.

| Land | Bestellwert | Genehmigung notwendig? |
|---|---|---|
| EMEA | < 100.000 | FALSE |
| | >= 100.000 | TRUE |
| APJ | < 70.000 | FALSE |
| | >= 70.000 | TRUE |
| AMER | < 150.000 | FALSE |
| | >= 150.000 | TRUE |

***Abbildung 113: Entscheidungstabelle für den vereinfachten Bestellprozess***

Je nach Herkunftsregion der Bestellung existieren unterschiedliche Bestellwertgrenzen, ab denen eine Genehmigung notwendig wird. Diese könnten nun, abhängig von der Geschäftsentwicklung, flexibel von fachlichen Experten zeitnah verändert werden. Dadurch liegt die Hoheit über den Geschäftsablauf wieder bei den Fachabteilungen.

Die Einhaltung von Bedingungen während der Prozessausführung gestaltet sich aufgrund der statischen Prozessflussmodellierung schon schwieriger. Gemäß einleitender Definition einer Bedingungsregel, handelt es sich dabei um Ausdrücke, die zu jedem Zeitpunkt erfüllt sein müssen. Graml, Bracht und Spies weisen darauf hin, dass

sich diese Bedingungen zusätzlich auf einen bestimmten Geltungsbereich beziehen, wie beispielsweise für eine bestimmte Sequenzfolge. Natürlich ließe sich die Einhaltung der Bedingungen nach einem ausgeführten Prozessschritt durch das Einfügen entsprechender Validierungsaufgaben gewährleisten. Allerdings könnte die Gültigkeit lediglich für den Überprüfungszeitpunkt sichergestellt werden. Nachfolgende Veränderungen der Daten und dadurch verursachte Verletzungen der Bedingung würden auf diese Weise nicht erfasst. Um auch dies sicherzustellen, bliebe nur das Hinzufügen von Validierungen nach jedem Prozessschritt, was natürlich nicht praktikabel ist. Offensichtlich handelt es sich bei derartigen Prüfungen, die sich über ganze Prozessabschnitte erstrecken, um klassische Cross Cutting Concerns, wie sie auch in der aspektorientierten Programmierung anzutreffen sind (Graml, Bracht, Spies 2008). In ihrer Lösung des Problems sehen sie in den BPMN-Modellen ausgewiesene Positionen vor, an denen die Geschäftsregeln aufgerufen werden, um die Einhaltung der Bedingungen sicherzustellen. In ihrer konkreten Umsetzung haben sie den aus dem BPMN-Modell erzeugten BPEL-Code durch XSLT-Transformationen nachträglich bearbeitet und automatisch Aufrufe an den Regelservice vor und nach jeder normalen Prozessaktivität hinzugefügt.

Dieses Verhalten lässt sich auf Basis BPMN 2.0 nun eleganter lösen. Dazu bietet sich die Verwendung von Ereignis-Teilprozessen an, wobei als Startereignis für eben diesen Ereignis-Teilprozess das Bedingungsereignis verwendet wird. Abbildung 114 veranschaulicht diesen Sachverhalt.

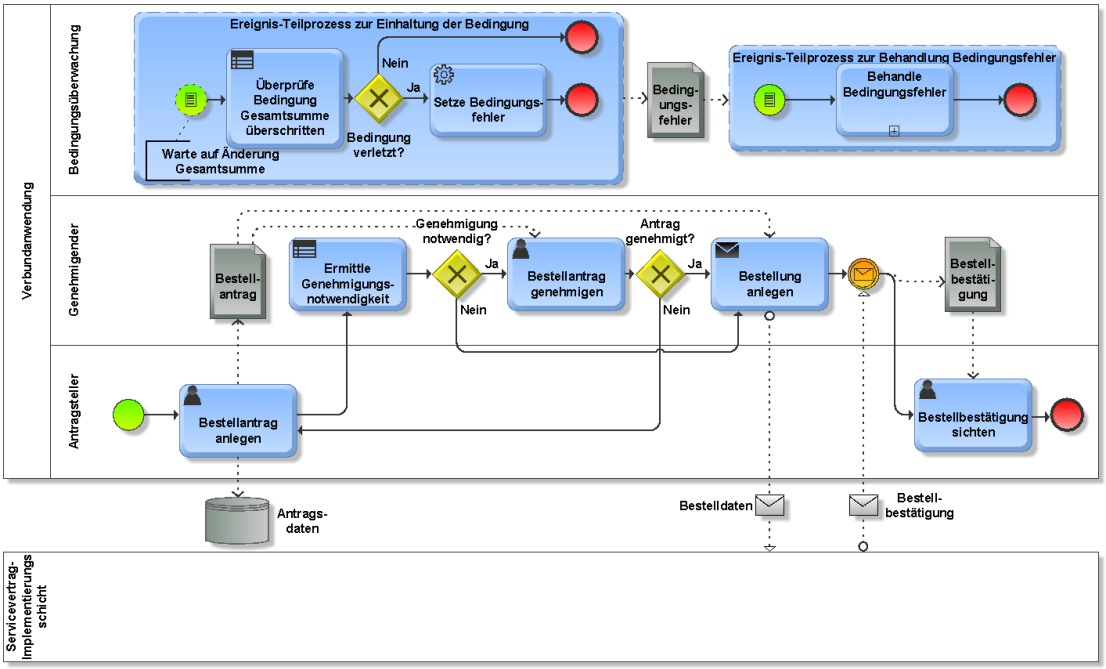



***Abbildung 114: Gewährleistung von Bedingungen über den Einsatz von Ereignis-Teilprozessen***

Der Hauptprozess wird von einem nicht unterbrechenden Ereignis-Teilprozess überwacht, der immer dann aktiviert wird, wenn sich die zu überwachende Variable (oder Variablen) ändert (ändern), beispielsweise die Gesamtsumme der Bestellung. Ist dies der Fall, so wird die Regel zur Einhaltung der Bedingung aufgerufen. Im obigen Modell könnte dies die Einhaltung einer bestimmten Gesamtsumme sein. Beispielsweise könnte die Regel aufgrund von Kunden- und Kreditwürdigkeitsdaten eine Obergrenze für die Bestellung festlegen, die nicht überschritten werden darf. Ist die Bedingung *Gesamtsumme* ***nicht*** *überschritten* erfüllt, so beendet sich der Teilprozess stillschweigend und der Hauptprozess kann mit seinem Prozessfluss fortfahren. Stellt hingegen die Regel eine Bedingungsverletzung fest, so wird ein Datenobjekt aktualisiert, auf dessen Veränderung wiederum ein zweiter Ereignis-Teilprozess wartet, der diesmal allerdings mit einem unterbrechenden Bedingungs-Startereignis beginnt. Dadurch wird die Beendigung des Hauptprozesses erzwungen und eine Fehlerbehandlung eingeleitet. Aufgrund der Erweiterung von BPMN 2.0 um Ereignis-Teilprozesse werden derartige Szenarien im Zusammenspiel mit Regelwerken nun standardmäßig unterstützt. Eine Nachbearbeitung in Form von XSLT-Transformationen ist nicht mehr notwendig.

Bleibt die Behandlung von Prozessregeln, die die Steuerung von Prozessfragmenten übernehmen. Die wesentliche Idee beruht darauf, bestimmte Prozessteile vorab zu modellieren, die jedoch nicht statisch miteinander verknüpft sind, sondern dynamisch zur Laufzeit über Prozessregeln zusammengefügt werden. Die Umsetzung dieser Idee ist genau das, was Graml, Bracht und Spies eingangs mit der Kombination von *flexibility by selection* und *flexibility by adaption* meinten. Dazu präsentieren sie in ihrem Papier zwei Pattern.

Im ersten Pattern wird eine Aktivität bzw. ein Teilprozess eines an sich statischen Prozesses austauschbar gehalten. Erst zur Laufzeit wird aufgrund einer Prozessregel bestimmt, welche konkrete Aktivität bzw. welcher konkrete Teilprozess die gewünschte Funktionalität zu erbringen hat. Als Beispiel führen die Autoren die Auslieferung eines Produktes als austauschbaren Teilprozess an, der je nach Land, in das die Lieferung zu erfolgen hat, unterschiedlich ausfällt, wobei sich die Geschäftsregel für die Auswahl des passenden Teilprozesses verantwortlich zeichnet. Die Selektionskriterien für den zu aktivierenden Teilprozess lassen sich aufgrund der Externalisierung als Regeln auch noch zur Laufzeit anpassen. Zudem können neue Teilprozesse definiert und eingesetzt werden, lange nachdem der Hauptprozess gestartet wurde. Allerdings beschränkt sich dieses Pattern lediglich auf einen wohldefinierten Aspekt innerhalb eines statischen Prozesses, der austauschbar gehalten wird. Die auszutauschenden Teilprozesse/Aktivitäten müssen zudem zwingend dieselbe Schnittstelle implementieren, damit sie aus dem statischen Prozess heraus aufgerufen werden können.

In realen Szenarien, insbesondere bei langlaufenden Prozessen, wird jedoch weitaus mehr Flexibilität verlangt: einige Prozessteile werden immer relativ konstant modellierbar bleiben, andere Fragmente müssen unter bestimmten Umständen ausgelassen werden, andere wiederum müssen zusätzlich hinzugenommen werden oder die Ausführungsreihenfolge ändert sich. Um auch diesen Anforderungen gerecht zu werden, schlägt das zweite Pattern eine dynamische Konfiguration von zu einander *unabhängigen* Prozessfragmenten zur Laufzeit über Geschäftsregeln vor. Die einzelnen Prozessfragmente repräsentieren dabei die Prozessanteile am Gesamtprozess, von denen angenommen werden kann, dass sie sich kaum ändern. Basierend auf dem Prozesskontext, der über die Prozessfragmente hinweg gehalten wird, und die Historie der bereits ausgeführten Teilprozesse, wird von dem Regelwerk bestimmt, welches

das nächste auszuführende Fragment sein soll. Die Autoren veranschaulichen die regelgesteuerte Komposition von Prozessfragmenten mit Abbildung 115.

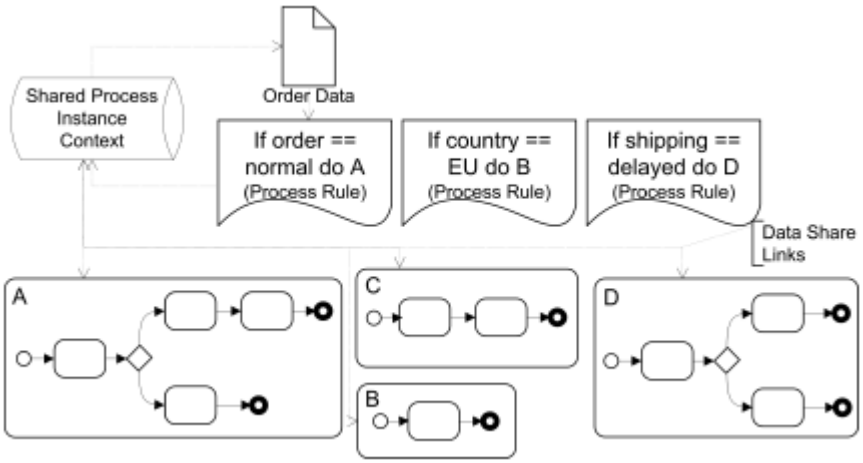



***Abbildung 115: Komposition von Prozessfragmenten (aus Graml, Bracht, Spies 2008)***

Letztendlich findet auch hier eine Umsetzung der Idee der losen Kopplung statt. Zur Implementierung dieses Patterns setzen Graml, Bracht und Spies einen zentralen Dispatcher-Prozess ein, der in einer Schleife das Regelwerk zur Bestimmung des nächsten Teilprozesses aufruft, um unmittelbar danach das derart ermittelte Prozessfragment auszuführen. Dies setzt allerdings voraus, dass sämtliche Teilprozesse auf den gemeinsam genutzten Prozesskontext abgestimmt sind. Nützliche Teilprozesse, die im Rahmen anderer Projekte entwickelt wurden und demnach mit hoher Wahrscheinlichkeit auf anderen Daten basieren, lassen sich mit dieser Methode nicht wiederverwenden. Allerdings lässt sich auch dies relativ einfach lösen, indem jedes Prozessfragment seine Funktionalität als Web Service anbietet und die Daten, auf denen das Fragment agiert, per Schnittstelle zur Verfügung gestellt bekommt. Der zentrale Dispatcher-Prozess ruft dann die Prozessfragmente nicht mehr direkt auf, da er nicht wissen kann, welche konkrete Schnittstelle jedes Prozessfragment implementiert, sondern übergibt die Informationen über das nächste auszuführende Prozessfragment zusammen mit den Daten des Prozesskontextes in Form einer Nachricht an einen ESB. Dieser kann nun durch seine Mapping-Funktionalität die Daten des Dispatcher-Prozesses auf die Daten des aufzurufenden Prozessfragments abbilden und somit auch Teilprozesse involvieren, die ursprünglich für ein anderes Szenario entwickelt wurden. Die Teilprozesse wiederum stellen ihre Ergebnisse dem ESB ebenfalls in Form

einer Nachricht zur Verfügung, so dass dieser dann wieder auf die Schnittstelle des Dispatcher-Prozesses mappen kann. Zusammengehalten wird das Gesamtszenario durch eine gemeinsame Prozess-ID, die bei Start des Dispatcher-Prozesses generiert wird. Der Dispatcher-Prozess wird pro Geschäftsvorfall instanziiert. Soll also im Gesamtszenario ein Bestellprozess abgewickelt werden, so ergibt sich ein Dispatcher-Prozess pro Bestellvorgang. Das Regelwerk selbst wird dann zu einer Art Zustandsautomat, der aufgrund der bereits abgewickelten Teilprozesse und dem Stand des Prozesskontextes entscheidet, wie weiter zu verfahren ist. Die folgende Abbildung 116 verdeutlicht das Verfahren.

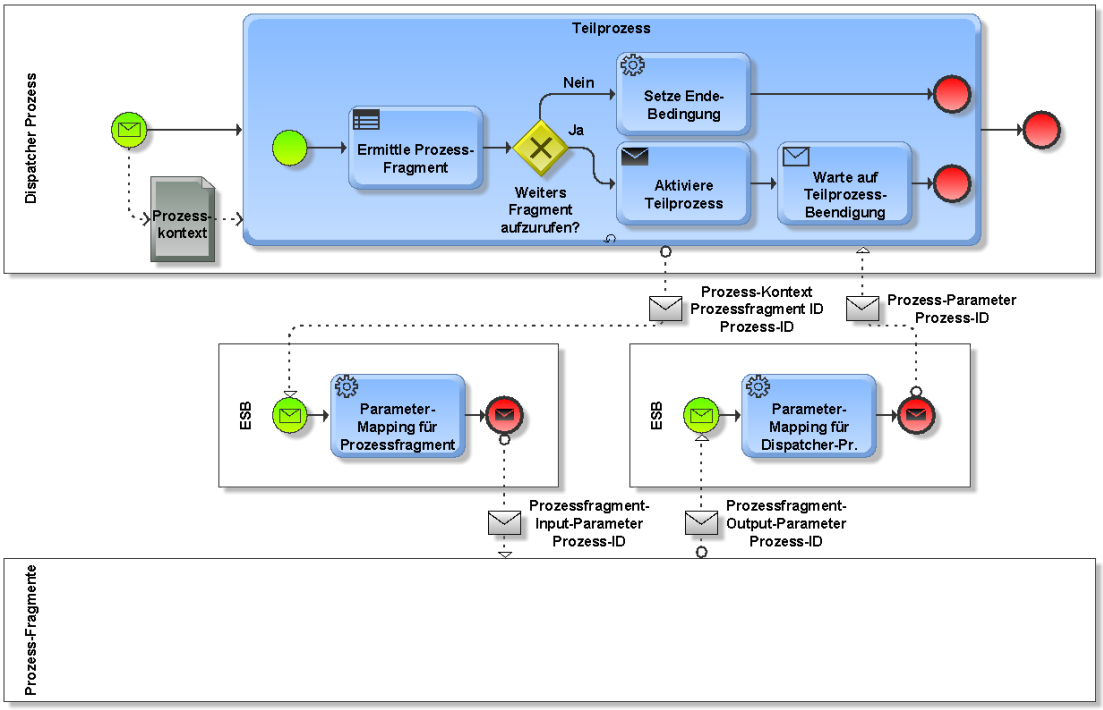



***Abbildung 116: Regelbasierte Verknüpfung von Prozessfragmenten über einen ESB***

Der Dispatcher-Prozess wird durch eine initiale Nachricht gestartet. In einem Schleifen-Teilprozess erfolgt die Prozessfragment-Ermittlung, der Prozessfragment-Aufruf sowie das Warten auf dessen Beendigung. Diese Schleife wird abgeschlossen, sobald die regelbasierte Prozessfragment-Ermittlung den Endzustand erkannt hat. Im Modell ist das Setzen der Endbedingung am *Nein*-Gate des exklusiven Oder-Gateways zu erkennen. Der Einsatz des Regelwerks befindet sich gleich zu Beginn des Teilprozesses durch Verwendung der in BPMN 2.0 neu eingeführten Geschäftsregel-Aufgabe. Eine vereinfachte Entscheidungstabelle zur Ermittlung der auf-

zurufenden Prozessfragmente sowie die Festlegung des jeweiligen Folgezustands ist in Abbildung 117 dargestellt.

| Zustand | Region | Prozessfragment | Folgezustand |
|---|---|---|---|
| Start | EMEA | Auftragsgenehmigung_EMEA | 1 |
| Start | US | Auftragsgenehmigung_US | 1 |
| Start | APJ | Auftragsgenehmigung_APJ | 1 |
| 1 | EMEA | Zollbehandlung_EMEA | 2 |
| 1 | US | Zollbehandlung_US | 2 |
| 1 | APJ | Zollbehandlung_APJ | 2 |
| 2 | EMEA, US, APJ | - | Ende |

***Abbildung 117: Entscheidungstabelle zur Ermittlung der aufzurufenden Prozessfragmente***

Die Entscheidungstabelle ist in zwei Hälften aufgeteilt: die Spalten *Zustand* und *Region* reflektieren die Eingabeparameter der Geschäftsregel und die Spalten *Prozessfragment* und *Folgezustand* die Ausgabeparameter. Da das Regelwerk direkt mit dem Dispatcher-Prozess verwoben ist und auch nur für das aktuell abzuwickelnde Geschäftsszenario einen Sinn ergibt, ist es unnötig, die Regel als Web Service bereitzustellen, da dessen Wiederverwendung eher unwahrscheinlich ist. Bei der Schnittstelle zwischen Prozess und Regelwerk ist es von großer Bedeutung, dass die Regel zur Entscheidungsfindung Zugang zum gesamten Prozesskontext besitzt, auch wenn zunächst nur ein Bruchteil der Daten des Kontextes für eine Entscheidung herangezogen wird. Dadurch kann das Regelwerk zu einem beliebigen späteren Zeitpunkt um neue Spalten oder neuen Regeln ergänzt werden, sofern dies notwendig wird. In der Beispieltabelle wird nach der jeweiligen Region, aus der der Aufruf des Prozesses erfolgte, entschieden, welches Prozessfragment zu aktivieren ist (Spalte *Prozessfragment*) und wie danach weiter verfahren wird (Spalte *Folgezustand*). Offensichtlich werden Auftragsgenehmigungs- und Zollabwicklungsprozesse je nach Region unterschieden.

Im *Ja*-Zweig des Gateways aus Abbildung 116 erfolgt der eigentliche Aufruf der Prozessfragmente über den ESB. Als Übergabeparameter an den ESB fungieren dabei der Prozesskontext, die eindeutige Kennung zum Aufruf des korrekten Prozessfragments (*Prozessfragment ID*) sowie die eindeutige Kennung des aufrufenden Dispatcher-Prozesses (*Prozess ID*). Diese ID wird zur späteren Zustellung der Meldung über die Beendigung des Teilprozesses benötigt. Der ESB selbst übernimmt lediglich das Mapping der Daten auf die Eingabeparameter des gerufenen Prozessfragments.

Grundsätzlich muss jedes Prozessfragment sowohl bei den Eingangs- als auch bei den Ausgangsparametern die Prozesskennung des aufrufenden Prozesses als Schnittstellenbestandteil berücksichtigen. Nur so können Rückgaben über Korrelation korrekt zugeordnet werden. Über die Prozess ID des Dispatcher-Prozesses wird letztendlich die gesamte Prozesskette zusammengehalten und ermöglicht dadurch ein durchgängiges ganzheitliches Monitoring, was auch zur Fehlerermittlung von fundamentaler Bedeutung ist. Folglich muss das Prozessfragment auch die eindeutige Prozess ID nach dessen Beendigung an den ESB liefern, der schließlich nach dem Mapping der Prozessfragment-Ausgabeparameter auf die Parameter des Dispatcher-Prozesses die richtige Dispatcher-Instanz reaktiviert.

Es stellt sich zwangsläufig die Frage, ob die Regel-Behandlung auch direkt in den ESB verlegt werden könnte und sich somit ein expliziter Dispatcher-Prozess erübrigt. Es ergäbe sich dadurch eine eher föderalistische, dezentrale Lösung des Problems ohne einen steuernden Zentralprozess. Dies ließe sich sicherlich implementieren, allerdings muss auch in einem solchen Fall der Zustand in irgendeiner Form berücksichtigt werden. Die eigentliche Aufgabe eines ESB's liegt in der Entgegennahme und Weiterleitung von Nachrichten unter Berücksichtigung typischer Aufgaben wie Routing und Mapping. Die Behandlung von Zuständen müsste also explizit eingebracht werden, z.B. durch Aufruf einer Persistierung durch JPA. Soll auf diese explizite Speicherung verzichtet werden, so sind alternativ alle Prozessdaten komplett von Prozessfragment zu Prozessfragment durchzuschleusen. Ein eher unwahrscheinliches Unterfangen, da die Wiederverwendbarkeit der Prozessfragment zusätzlich eingeschränkt würde. Betrachtet man einen Gesamtprozess als Summe der aufgerufenen Prozessfragmente, so wird ein Gesamtprozess für ein Anlagengeschäft auf andere Daten arbeiten als ein Gesamtprozess für ein Projektgeschäft. Beide Gesamtprozesse werden aber sicherlich das Prozessfragment *Rechnungsabwicklung* wiederverwenden wollen. Dessen Schnittstelle sollte allerdings nicht von den beiden Verwendungsarten abhängig sein, was aber bei der Weitergabe des jeweiligen kompletten Kontextes nicht gewährleistet wäre. Von daher scheidet diese Möglichkeit aus.

Folglich wird eine Art Prozess zur Zustandsverwaltung benötigt. Diesen in Form eines koordinierenden Dispatcher-Prozesses zentral durchzuführen ist aus Sicht der Nachvollziehbarkeit des Prozessablaufes, des Monitorings oder auch der Fehleranaly-

se sinnvoll. Eine Regel-Behandlung direkt im ESB für den hier geschilderten Anwendungsfall erscheint daher nicht angebracht.

Dieses Verfahren eignet sich auch für das in Abschnitt 4.5 diskutierte Vorgehen zur dynamisch-maschinellen Komponentenauswahl. In der Entscheidungstabelle aus Abbildung 117 werden anstelle der Prozessfragmente lediglich die passenden Komponenten zugewiesen. Bei Komponenten mit unterschiedlichen Schnittstellen sorgt der ESB dann für eine entsprechende Vermittlung. Weitere Details zur Konfiguration dynamischer Anwendungssysteme aus Komponenten wurden in Abschnitt 4.5 besprochen.

Doch damit sind die Einsatzmöglichkeiten von Geschäftsregeln in Verbundanwendungen bei weitem noch nicht erschöpft. Im Folgenden werden kurz weitere mögliche Anwendungsfälle diskutiert.

Im ersten Fall wird die Idee des Zustandsautomaten erneut aufgegriffen, diesmal jedoch, um die Reihenfolge von Prozessschritten zu variieren. Sind die einzelnen Schritte eines Prozesses zwar bekannt, die Reihenfolge aber unterschiedlich oder können einzelne Schritte sogar entfallen, so bietet sich einmal mehr der Einsatz von Entscheidungstabellen an. Die wesentliche Idee lässt sich an den Prozessschritten Bestellen-Essen-Bezahlen für unterschiedliche Restaurantarten beispielhaft erläutern (siehe auch Grimm, Speck, Stiehl 2009). In einem klassischen Restaurant wird zunächst bestellt, anschließend gegessen und zum Abschluss bezahlt. In einem McDonalds-Restaurant wird hingegen bereits nach der Bestellung bezahlt und erst dann gegessen. Bei einem Buffet wird in der Regel ein Pauschalpreis bezahlt und danach gegessen. Der Bestellschritt entfällt völlig. Um einen Prozess für alle drei Restaurants zu implementieren, wird die benötigte Flexibilität durch den Einsatz von Regeln erreicht. Die beiden nachfolgenden Abbildungen zeigen die für ein solches Szenario benötigte Entscheidungstabelle und den dazugehörigen Prozessablauf.

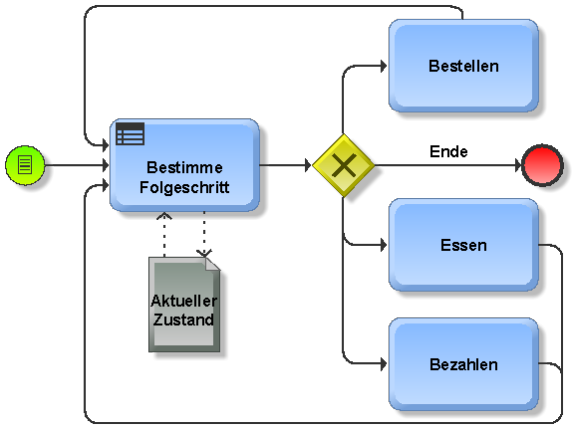



*Abbildung 118: Variation von Prozessschritten unter Verwendung von Regeln*

| Zustand | Restaurant | Prozessschritt | Folgezustand |
|---|---|---|---|
| Start | Klassisch | Bestellen | 1 |
| Start | McDonalds | Bestellen | 1 |
| Start | Buffet | Bezahlen | 1 |
| 1 | Klassisch | Essen | 2 |
| 1 | McDonalds | Bezahlen | 2 |
| 1 | Buffet | Essen | Ende |
| 2 | Klassisch | Bezahlen | Ende |
| 2 | McDonalds | Essen | Ende |
| Ende | * | Ende | |

*Abbildung 119: Entscheidungstabelle zur Variation von Prozessschritten*

Nach Erfüllung der Startbedingung, in diesem Szenario beispielsweise das Betreten des Restaurants durch einen Kunden, wird zunächst der auszuführende Prozessschritt und dabei gleichzeitig der neue Zustand durch das Regelwerk ermittelt. Anschließend kommt es zur Ausführung des festgelegten Schrittes, ehe der nächste Schleifendurchlauf wieder mit der Folgeschrittermittlung startet. Dieses Pattern lässt sich auf eine Vielzahl gleichgearteter Szenarien anwenden. Der Nachteil liegt lediglich in der zuvor fix festgelegten Auswahl der Prozessschritte. Gänzlich neue Schritte lassen sich mit diesem Verfahren nicht hinzufügen. Dazu müsste erneut auf die lose gekoppelte Variante über einen ESB zurückgegriffen werden. Allerdings ist es eine berechtigte Frage, ob der nicht ganz unerhebliche Implementierungsaufwand für Einzelschritte sinnvoll ist. In den meisten Fällen wird die Reihenfolgeflexibilität, wie soeben beschrieben, ausreichend sein. Es kann jederzeit zur Laufzeit eine beliebige Reihenfolge konfiguriert werden. Eine generischere Variante dieser Idee zeigt das Modell

aus Abbildung 120 mit der dazugehörigen Entscheidungstabelle in Abbildung 121. Darin wird eine länderspezifische Ausführungsreihenfolge der Aktivitäten festgelegt.

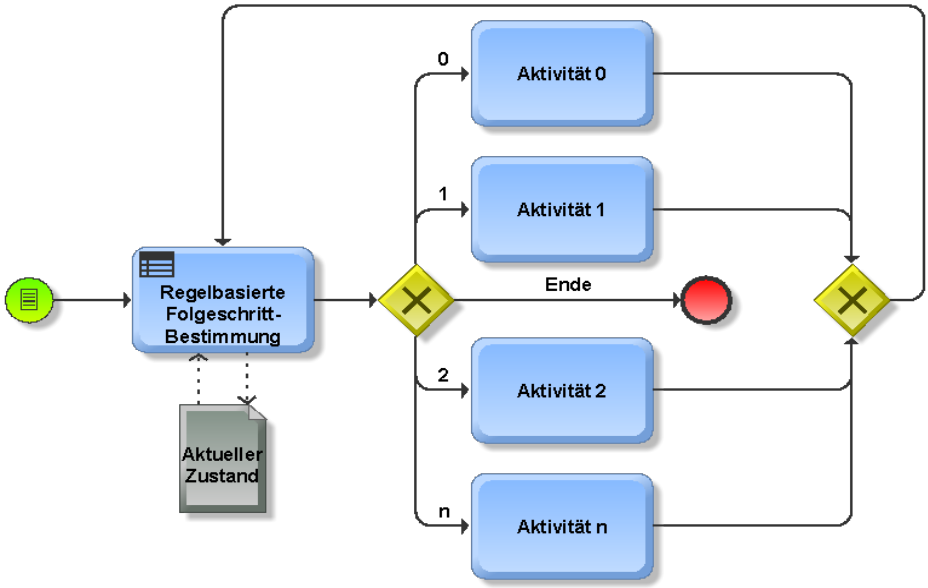



*Abbildung 120: Generische Prozessvariante zur Steuerung von Prozessschritten*

| Land | Aktueller Zustand | Folgezustand |
|---|---|---|
| DE | Start | 0 |
| | 0 | 1 |
| | 1 | 2 |
| | 2 | n |
| | n | Ende |
| US | Start | 0 |
| | 0 | 2 |
| | 2 | n |
| | n | Ende |
| Fr | Start | 0 |
| | 0 | 2 |
| | 1 | n |
| | 2 | 1 |
| | n | Ende |

*Abbildung 121: Entscheidungstabelle zur generischen Steuerung von Prozessschritten*

Je nach Land ergibt sich eine unterschiedliche Ausführungssequenz: für Deutschland die Folge 0 → 1 → 2 → n und für die USA die Folge 0 → 2 → n (Start- und Endzustand nicht mit aufgeführt). Schritte können ausgelassen werden, wie dies soeben für die USA erläutert wurde und das Beispiel für Frankreich zeigt, wie Schritte

in anderer Folge aufgerufen werden können (Sequenzfolge: 0 → 2 → 1 → n). Das Hinzufügen/Löschen von Ländern lässt sich durch den Einsatz von Regelwerken ebenso einfach realisieren wie die Veränderung von Schrittfolgen. Dies sind die entscheidenden Argumente, die für den Einsatz von Geschäftsregeln sprechen, unter Berücksichtigung der o.g. Einschränkungen der fix vorgegebenen Aktivitäten.

Regeln finden auch bei Personalfragen während der Prozessausführung ihre Anwendung. Dabei geht es darum, wer bestimmte Schritte ausführen darf, wer von der Ausführung ausgeschlossen wird oder wer als Administrator Zugang zu den Prozessen hat. Hier wird auch eine Schwäche von BPMN offenbar: zur Modellierung von Rollen werden in der BPMN für gewöhnlich die Schwimmbahnen verwendet. Dabei können bestimmte Sachverhalte nicht oder nur durch Ergänzung von Kommentaren zum Ausdruck gebracht werden. Auch die Verbindung zu Organigrammen ist nicht vorhanden. BPMN will diese Lücken auch gar nicht schließen. Es ist vielmehr die Strategie der OMG, derartige Sachverhalte durch andere Standards abzudecken und diese zukünftig mit der BPMN in Verbindung zu bringen. Als Beispiel für diese offensichtliche BPMN-Schwäche ist in Abbildung 122 ein Prozessfragment aus Allweyer 2010a wiedergegeben.

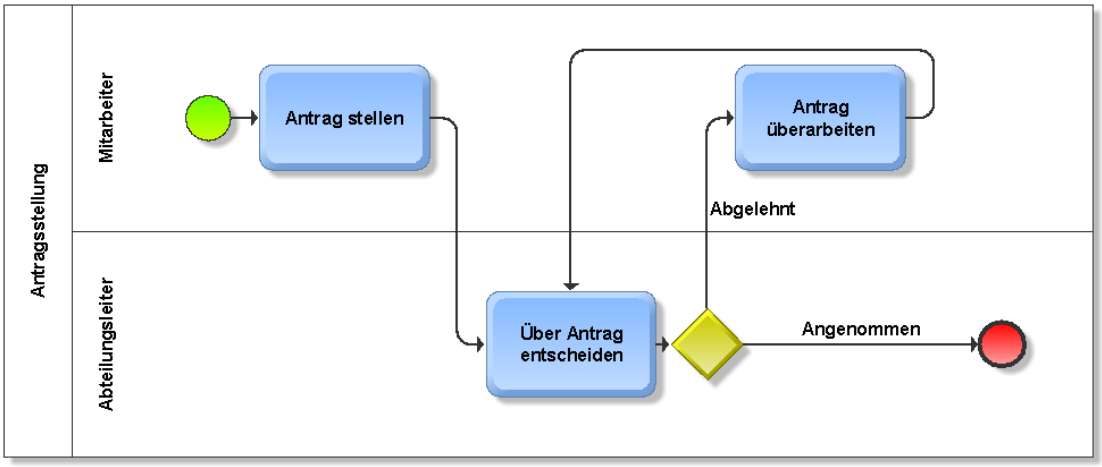



***Abbildung 122: Einfacher Prozess zur Antragsstellung (nach Allweyer 2010a)***

Der Schritt *Antrag stellen* dieses Antragsstellungsprozesses kann durch einen beliebigen Mitarbeiter ausgeführt werden, während dies für die Aufgabe *Antrag überarbeiten*, obwohl in derselben Schwimmbahn platziert, nicht gilt. Offensichtlich soll diese

Aktivität nur durch denjenigen Mitarbeiter ausgeführt werden, der den Antrag im ersten Prozessschritt ursprünglich gestellt hatte. Auch die *Abteilungsleiter*-Schwimmbahn ist nicht unproblematisch, denn nicht jeder beliebige Abteilungsleiter soll die Aufgabe *Über Antrag entscheiden* ausführen können, sondern lediglich der Leiter der Abteilung, zu dem der Antragsteller gehört. Derartige Konstellationen gilt es immer wieder zu berücksichtigen und in den Griff zu bekommen. Regeln bieten hier eine vernünftige Alternative im Zusammenspiel mit Benutzerverwaltungssystemen. Die eigentliche Benutzerzuweisung wird folglich zur Laufzeit ermittelt und ist nicht statisch mit den Schwimmbahnen verbunden. Die Schwimmbahnen geben eine gewisse Orientierung, welche Personengruppe für welche Aktivitäten verantwortlich ist. Wie diese allerdings zur Laufzeit konkret ermittelt wird, obliegt dem Regelwerk, so dass auch hier zur Laufzeit korrigierend eingegriffen werden kann. Dies wird immer dann notwendig sein, wenn kritische Prozesse aufgrund von Fehlzeiten der beteiligten Personen (Urlaub, Krankheit) verzögert werden und schnelle Anpassungen erforderlich sind. Dabei werden in den Regeln keine Personen direkt eingetragen, da dies zu statisch wäre. Stattdessen wird auf Gruppen und Rollen in klassischen Benutzerverwaltungssystemen wie LDAP-basierte Systeme, Datenbanken oder SAP-Systeme verwiesen. In diesen Systemen sind auch die Relationen zwischen den Personen hinterlegt. Sind also spezielle Beziehungen (z.B. Vorgesetzter) zu implementieren, wie dies im obigen Szenario verlangt wird, so muss das Benutzerverwaltungssystem durch Bereitstellung eines APIs unterstützen. Beispielsweise müssen Vorgesetzte entlang der Organisationshierarchie ermittelbar sein. Entweder sind diese Aufrufe direkt aus der Regelmaschine heraus möglich und die Geschäftsregeln liefern konkrete Namen oder die Regel gibt entsprechende Anweisungen an den aufrufenden Prozess zurück und dieser ermittelt die Personengruppe durch Aufrufe an die Benutzerverwaltung. Dabei wird eine enge Verzahnung zwischen Prozess und Geschäftsregeln vorausgesetzt: der Prozess muss also wissen, wie er eine Regelvorgabe wie *Manager ermitteln* in Aufrufe an die Benutzerverwaltung umzusetzen hat, damit das gewünschte Verhalten erreicht wird.

Das obige Beispiel des Antragsstellungsprozesses tangiert ein weiteres Problem: die geeignete Modellierung des 4-Augen-Prinzips. Das in Abbildung 122 dargestellte Modell zeigt die klassische Lösung: der Antrag wird von seinem Vorgesetzten genehmigt, wobei Antragsteller und Manager in zwei separierten Schwimmbahnen darge-

stellt werden. Doch was ist, wenn dieses Prinzip aufgrund von firmeninternen Änderungen angepasst werden muss? Neben dem direkten Vorgesetzten soll bei bestimmten Konstellationen, wie beispielsweise ein erhöhter Antragswert, nicht nur der direkte Vorgesetzte sondern auch dessen Manager involviert werden. In manchen Fällen soll auch nicht der direkte Vorgesetzte involviert werden, sondern derjenige Verantwortliche in der Hierarchie, der über eine bestimmte Unterschriftsberechtigung verfügt. Es macht keinen Sinn, jedes Mal den Prozess diesen Gegebenheiten anzupassen. Stattdessen muss die Flexibilität erneut aus einer Kombination von Geschäftsregeln und Prozess erfolgen, womit sich einmal mehr die Aussage bestätigt, dass durch den Einsatz von Regeln die eigentlichen Geschäftsprozesse stabil gehalten werden können. Eine mögliche Lösung des Problems zeigt Abbildung 123.

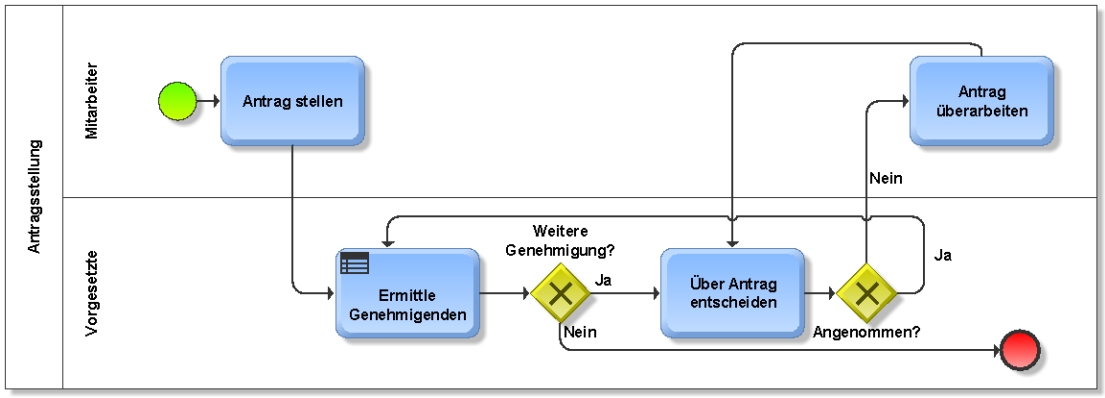



***Abbildung 123: Implementierung von mehrfachen Genehmigungen durch Einsatz von Regeln***

Die Genehmigung wird nun in einer regelbasierten Schleife abgewickelt. In dem Modell wird angenommen, dass aus der Regel heraus das API der Benutzerverwaltung gerufen und der Verantwortliche/die Verantwortlichen ermittelt und der Schwimmbahn zugewiesen werden können. Dadurch kann die Aufgabe *Über Antrag entscheiden* von der richtigen Person übernommen werden. Egal wie viele Ebenen zukünftig berücksichtigt werden müssen, dieser Ansatz lässt beliebige Kombinationen zu, ohne dass der Prozess dafür angepasst werden müsste.

Bei den bereits diskutierten Prozess-Pattern wurde auch immer der zeitliche Aspekt berücksichtigt: die Ausführung von Einzelschritten oder Schrittsequenzen zusammengefasst in Teilprozessen wurden mit angehefteten Timer-Ereignissen auf ihre Ausführungszeiten hin überwacht und bei Überschreitung eine passende Ausnahmesequenz

aktiviert. Auch für Benutzeraufgaben können unterschiedlichste Zeiten definiert werden. So kann der Zeitpunkt definiert werden, bis zu dem die Aufgabe begonnen werden oder bis zu welchem Zeitpunkt sie beendet sein muss. Werden die Zeiten überschritten, so sorgt ein Timer-Ereignis für die Aktivierung der Ausnahmebehandlungs-Sequenzfolge.

Für manche Aufgaben kann auch die Freigabe durch einen sogenannten Aktivierungs-Timer zeitlich vorgegeben werden: so können sich Interessanten an einer Umfrage anmelden, die dazugehörige Umfrage soll allerdings erst zu einem bestimmten Zeitpunkt starten. Ist dieser Zeitpunkt erreicht, wird der dazugehörige Schritt freigegeben und das Umfrageformular an die Interessenten verschickt. Durch die zeitliche Vorgabe wird also der gleichzeitige Start der Kampagne gewährleistet.

Ein letztes Beispiel für die Verwendung von Zeitvorgaben in Zusammenhang mit Regeln ist die Definition eines Verfallstermins: ist bis zu einem vorab definierbaren Zeitpunkt eine Aufgabe noch nicht bearbeitet worden, so soll sie verfallen, da die Eingabedaten für den weiteren Prozessablauf nicht mehr relevant sind. Bezugnehmend auf die Umfragekampagne bedeutet die Vorgabe eines Verfallstermins, dass Antworten von Teilnehmern, die die Daten erst nach einem bestimmten Datum übermitteln, bei der Auswertung nicht mehr berücksichtigt werden.

Für all die genannten Beispiele kann wiederum die Logik zur Ermittlung der jeweiligen Zeitpunkte in Regeln ausgelagert und somit zur Laufzeit anpassbar gemacht werden. Weitere Einsatzgebiete von Regeln sind Konditionen für Ausnahmen (unter welchen Bedingungen soll welche Ausnahme ausgelöst werden?) sowie Abbruchbedingungen für Schleifen (welche Konstellation muss eintreten, damit eine Schleife beendet wird?). In dem nun folgenden Abschnitt werden weitere Szenarien speziell für technische Prozesse besprochen.

### *5.7.2 Einsatz von Geschäftsregeln in technischen Prozessen*

Der Einsatz von Regeln bietet sich auch für die Servicevertrag-Implementierungsschicht an, in der bekanntlich technische Integrationsprozesse realisiert werden. Bei der Verarbeitung einer Nachricht müssen in der Regel mehrere Entscheidungen getroffen werden:

- Wer ist/sind der Empfänger/die Empfänger der Nachricht?

- Welche Schnittstelle/n erwartet/erwarten der/die Empfänger?
- Wie werden die einzelnen Felder der Nachricht in das/die Zielformat/e transformiert?
- Ist die Eingangsnachricht unmittelbar weiterzuleiten oder muss auf weitere Nachrichten gewartet werden? Wenn gewartet werden soll, wie lange soll gewartet werden, in welcher Reihenfolge sollen die Nachrichten verschickt werden und wie ist in Fehlerfällen zu reagieren?
- Muss die Nachricht weitere Bearbeitungsschritte durchlaufen, ehe sie dem Empfänger/den Empfängern zugestellt werden kann und wenn ja, in welcher Reihenfolge soll dies geschehen?
- Über welchen Kanal wird welche Nachricht an den Empfänger gesendet?

Diese Liste, die keinen Anspruch auf Vollständigkeit erhebt, zeigt allerdings, welch vielfältige Einsatzmöglichkeiten es für Regeln gibt, denn für die obigen Entscheidungen bieten sich Regelwerke als Implementierungsmittel an. Die aufgeführten Fragestellungen sind klassische Integrationsprobleme, zu deren Lösung Hohpe und Woolf in ihrem Enterprise Integration Patterns-Buch (Hohpe & Woolf 2004) vielfältige Lösungspattern entwickelt haben. Sie sollen an dieser Stelle nicht wiederholt werden. Stattdessen wird an ausgewählten Beispielen aus obigem Spektrum gezeigt, wie konkret ein Regelwerk eingesetzt werden kann. Es bieten sich dazu die Bereiche Empfänger-Ermittlung, Schnittstellenermittlung, Mapping sowie Kanalermittlung und Nachrichtenbehandlungslogik an.

Trifft eine Nachricht in der Servicevertrag-Implementierungsschicht ein, so sind zunächst einmal dessen Absender, die Eingangsschnittstelle, sowie der Kanal, über den die Nachricht eintraf, bekannt. Mit *Kanal* ist in diesem Zusammenhang die Art und Weise des Nachrichtentransports gemeint, also über welchen Weg ist die Nachricht übermittelt worden: per Email, JMS-Queue, Web Service-Aufruf, oder per Filetransfer. An dieser Stelle ist bereits der erste Einsatz von Regeln denkbar. Basierend auf den genannten Informationen lassen sich der oder die Empfänger ermitteln. Dafür könnten zudem der Inhalt der Nachricht einschließlich weiterer Metadaten wie die Nachrichtenlänge oder das Nachrichtenformat zur Entscheidungsfindung herangezogen werden. Sowohl Entscheidungstabellen als auch herkömmliche Wenn-Dann-Re-

geln eignen sich zur Umsetzung. Typische Beispiele hierfür sehen in etwa wie folgt aus:

- Wenn die Nachricht mit der Schnittstelle *OrderOut* über Email vom Absendersystem *ABC* eintrifft, dann ist der Empfänger *XYZ*.
- Wenn die Nachricht mit der Schnittstelle *InvoiceOut* eintrifft und das Feld *totalAmount* den Wert *1.000.000* in der Währung *Euro* überschreitet, so sind die Empfänger *LMN* und *UVW*.

Ist auf diese Weise der Empfänger ermittelt worden, steht bereits der nächste Regeleinsatz an, nämlich die Ermittlung des Zielschnittstellformats, also die Beantwortung der Frage, in welchem Format der ermittelte Empfänger die Nachricht erhalten soll. Auch hier können erneut die Informationen wiederverwendet werden, die bereits bei der Empfängerermittlung nützlich waren. Es bietet sich zudem eine Trennung in unterschiedliche Regelwerke zur Empfänger- und Schnittstellenermittlung an, da sich dadurch die Flexibilität des Gesamtsystems erhöht und jedes Regelwerk genau einen Aspekt erfasst (*separation of concerns*). Die Administration der Regelwerke wird durch diese Trennung zumindest dahingehend erleichtert, als das die einzelnen Regelwerke weniger Einträge umfassen. Allerdings leidet die Übersichtlichkeit, da getrennte Regelwerke gepflegt werden müssen und der Administrator nicht alles auf einen Blick erfassen kann. Dennoch dürfte die Flexibilitätssteigerung den Ausschlag für eine Trennung geben.

Beispiele für die Ermittlung der Zielschnittstelle sehen in etwa wie folgt aus:

- Wenn die Eingangsschnittstelle vom Format *OrderOut* ist und an den Empfänger *XYZ* gesendet werden soll, so ist das Zielformat *OrderInXYZ*.
- Wenn die Eingangsschnittstelle vom Format *InvoiceOut* ist, an den Empfänger *UVW* gesendet werden soll und das Feld *urgency* den Wert *critical* enthält, so ist das Zielformat *InvoiceInCriticalUVW*.

Zu diesem Zeitpunkt sind bei der Nachrichtenbearbeitung sowohl der/die Empfänger als auch das/die Zielformat/e bekannt. Nun muss die Nachricht aus dem Eingangs- in das Ausgangsformat umgesetzt werden, wofür Mapping-Regeln geeignet sind. Sie

legen fest, wie Ein- und Ausgangsfelder einander zugeordnet werden. Kerneinsatzbereich sind dabei die Werte-Mappings, also die Umwandlung von Werten, die nicht 1-zu-1 übernommen werden können. Die Felder der Schnittstellen repräsentieren zwar dieselben semantischen Informationen, allerdings werden sie wertmäßig unterschiedlich dargestellt. So liefert beispielsweise die Eingangsschnittstelle den Wert *1* für das Geschlecht *männlich* und den Wert *2* für *weiblich*. Die Ausgangsschnittstelle kann hingegen mit den Werten *1* und *2* nicht umgehen und braucht folglich eine für sie verständliche Darstellung wie die genannten Langtexte. Folglich ergeben sich Regeln der Form

- Wenn das Eingangsfeld *gender* den Wert *1* enthält, so erhält des Ausgangsfeld *geschlecht* den Wert *männlich*.

Für derart einfache Umwandlungen bieten sich aber auch Umwandlungstabellen statt Wenn-Dann-Regeln an, da sie für solche Fälle deutlich einfacher zu pflegen sind. Werden hingegen aufwändigere logische Ausdrücke oder gar umfassende Berechnungen zur Bestimmung eines Zielwerts benötigt, sind klassische Wenn-Dann-Regeln wieder das Mittel der Wahl. Ein typisches Beispiel hierfür ist die Ermittlung des Preises abhängig von einer Vielzahl von Parametern wie z.B. dem Besteller, der Bestellmenge, dem bestellten Produkt, dem Kundenstatus usw. Liefert die Eingangsschnittstelle all diese Daten jedoch ohne eine Preisinformation, die allerdings von der Ausgangsschnittstelle zwingend benötigt wird, so muss oftmals eine komplexe Logik durchlaufen werden, ehe der Preis ermittelt werden kann. Es ist daher sinnvoll, diese Logik als wiederverwendbares Regelwerk auszulagern.

Die Nachricht liegt nun versandfertig im Zielformat mit korrekt gefüllten Feldern vor. Das nächste Regelwerk bestimmt jetzt die Art und Weise, wie die Nachricht zum Empfänger transportiert werden soll. Dieser als *Kanalermittlung* bezeichnete Schritt legt fest, ob der Versand beispielsweise per JMS-Queue, per Web Service, per File-Transfer oder per Email zu erfolgen hat. Dies lässt sich aufgrund von Regeln wiederum feingranular steuern. So können selbst bei ein und derselben Ausgangsschnittstelle und demselben Empfänger aufgrund unterschiedlicher Feldinhalte unterschiedliche Kanäle zugewiesen werden. Die dazugehörigen Regeln haben dabei folgendes Aussehen:

- Wenn der Empfänger *UVW* die Nachricht *InvoiceInCriticalUVW* erhalten soll und das Feld *customerStatus* den Wert *gold* enthält, so ist der Ausgangskanal *Email* zu verwenden.
- Wenn der Empfänger *UVW* die Nachricht *InvoiceInCriticalUVW* erhalten soll und das Feld *customerStatus* den Wert *silver* enthält, so ist der Ausgangskanal *JMS* zu verwenden.

Damit ist die Nachrichtenbehandlung abgeschlossen. Sämtliche Informationen zum Versand liegen nun vor und die Nachricht kann...

- ...im korrekten Format
- ...mit korrekten Werten
- ...über den ermittelten Kanal
- ...an den richtigen Empfänger

verschickt werden. Nicht weniger als vier unterschiedliche Regelwerke waren notwendig, um dieses Ergebnis zu erreichen.

Als weiteres Beispiel für die Verwendung von Geschäftsregeln innerhalb technischer Integrationsprozesse soll eine bestimmte Nachrichtenbehandlungslogik implementiert werden. Diese tritt stets bei komplexen Nachrichtensequenzen auf, bei denen also die Integrationsschicht den korrekten Nachrichtenablauf zu überwachen hat. So werden z.B. bei einem vollständigen Auftragsbearbeitungsprozess folgende Nachrichten eine Rolle spielen:

- Bestellung anlegen/bestätigen/ändern/stornieren
- Verkaufsauftrag anlegen/ändern/stornieren
- Lieferung anlegen/ändern
- Warenbewegung anlegen/ändern
- Rechnung anlegen/ändern
- Zahlungseingang anlegen

Ist nun die Integrationsschicht für die korrekte Reihenfolge all dieser Nachrichten verantwortlich, muss per Regelwerk eindeutig definiert werden, wie zu verfahren ist, wenn die Nachrichten eben nicht in der erwarteten Sequenz eintreffen. Dies kann immer dann passieren, wenn es durch Ausfälle von Netzwerken und Rechnern zu Störungen kommt. Eine einfache Fallstudie verdeutlicht dies: für die Produktion benötigt ein Autohersteller dringend Material. Daher wird der Hersteller elektronisch eine Bestellung anlegen. Durch eine Störung im Netzwerk erreicht die Nachricht den Lieferanten allerdings nicht. Nachdem die Störung entdeckt wurde und eine kurzfristige Reparatur ausgeschlossen ist, ruft der Autohersteller kurzerhand den Lieferanten an. Dieser gibt die Daten direkt in sein Auftragserfassungssystem ein, worauf dieses wiederum automatisch eine Bestellbestätigungsnachricht versendet. Folglich erreicht die Integrationsschicht die Bestellbestätigung, obwohl der Bestellvorgang eigentlich mit einer Bestellung beginnen müsste. Durch Regeln lässt sich nun festlegen, wie zu verfahren ist: soll die Bestellbestätigung so lange zurückgehalten werden, bis die dazu passende Bestellung eintrifft (sie ist ja noch immer beim Hersteller gespeichert und wird, sobald die Leitung wieder repariert ist, von diesem auch wieder gesendet) oder soll die Bestellbestätigung weitergeleitet und die dann später eintreffende Bestellung ignoriert werden?

Erschwert wird diese Logik noch dadurch, dass manche Nachrichten unbedingt eintreffen müssen, andere wiederum optional sind oder dass spezielle Nachrichten genau einmal verschickt werden während andere Nachrichtentypen durchaus öfter auftreten können. All dies muss bei der Erstellung der Geschäftsregeln berücksichtigt werden. Als Ergebnis erhält man durch Kombination von Prozess und Geschäftsregeln einen Zustandsautomat, der für jede mögliche Situation entscheiden kann, wie weiter zu verfahren ist. Ein kleines Beispiel verdeutlicht eine mögliche Lösung. Dem Beispiel wird dabei die in Abbildung 124 dargestellte Entscheidungstabelle zugrunde gelegt.

| Zustand | N0 | N1 | N2 | N3 |
|---|---|---|---|---|
| 0000 | F, 0001 | H, 0010 | H, 0100 | H, 1000 |
| 0001 | Ungültig | F, 0011 | F, 0101 | H, 1001 |
| 0010 | F, 0011 | H, 0010 | H, 0110 | H, 1010 |
| 0011 | Ungültig | F, 0011 | F, 0111 | H, 1011 |
| 0100 | F, 0101 | C, 0110 | H, 0100 | H, 1100 |

***Abbildung 124: Ausschnitt der Entscheidungstabelle für komplexe Nachrichtensteuerungssequenzen***

In dem Beispiel sollen vier Nachrichten (N0 bis N3) verarbeitet werden. Der Zustandsautomat befindet sich initial im Zustand 0000. Dabei spielt die Darstellung des Zustands eine entscheidende Rolle: jede Stelle repräsentiert, binär kodiert und von rechts gelesen, welche Nachricht bereits eingetroffen ist. Ein 0 steht für „noch nicht eingetroffen“ und eine 1 für „bereits eingetroffen“. Somit bedeutet der Initialzustand 0000, dass noch keine Nachricht behandelt wurde. Zudem wird angenommen dass...

- ...die Startnachricht N0 genau einmal auftaucht,
- ...die Nachricht N1 eine optionale Nachricht ist
- ...die Nachrichten N0, N2 und N3 auftreten müssen

Die von dem Prozess auszuführende Aktion für die aktuell eingetroffene Nachricht ist nun in den Aktionsfeldern der Tabelle hinterlegt. Sie enthalten zum einen die Information, was mit der aktuellen Nachricht geschehen soll, kodiert durch die Buchstaben F, H, C für **F**orward (Weiterleiten), **H**old-Back (Zurückhalten) und **C**ancel (Ignorieren). Zum anderen wird der Folgezustand des Automaten hinterlegt.

Die erste Zeile der Tabelle repräsentiert demnach die Zustandsübergänge aus dem Initialzustand heraus. Bei Eintreffen der Startnachricht N0 kann diese sofort weitergeleitet werden (Forward). Das erste Bit der Zustandsdarstellung wird gesetzt, so dass sich als Folgezustand 0001 ergibt. Bei Eintreffen von N1 hingegen, kann diese noch nicht weitergeleitet werden (Hold-Back), da zwingend auf die Startnachricht N0 gewartet wird. Also wird lediglich der Folgezustand 0010 gesetzt, der für die eingetroffene Nachricht N1 steht. Dasselbe gilt für das Eintreffen der Nachrichten N2 und N3.

Die zweite Zeile der Tabelle betrachtet die Übergänge bei bereits eingetroffener Nachricht N0 (entspricht Zustand 0001). Ein weiteres Eintreffen der Nachricht N0 ist per Definition nicht erlaubt, daher der Eintrag *Ungültig* in dem dazugehörigen Akti-

onsfeld. Anders sieht es hingegen beim Eintreffen von N1 aus. Sie kann sofort verarbeitet werden und der Automat in den Zustand 0011 wechseln. Interessant ist auch das nächste Feld, das das Eintreffen von N2 repräsentiert. Obwohl N1 noch nicht eingetroffen ist, wird N2 gemäß Tabelleneintrag sofort weiterverarbeitet und der Automat nimmt den Zustand 0101 ein. Dies hängt damit zusammen, dass N1 als optionale Nachricht nicht notwendigerweise eintreffen muss. Daher kann N2 in der Tat sofort behandelt werden. Anders sieht es hingegen schon wieder in der letzten Tabellenspalte aus: bei Eintreffen von N3 kann diese nicht sofort abgearbeitet werden, da dazwischen die Pflichtnachricht N2 fehlt. Also ist es richtig, durch den Eintrag *H, 1001* N3 zurückzuhalten und lediglich in den Zustand 1001 zu wechseln. Auf diese Weise lässt sich obige Tabelle recht einfach entschlüsseln.

Ein letztes interessantes Feld ist in der letzten Zeile der Tabelle für den Zustand 0100 zu erkennen. Sie repräsentiert den Fall, dass bisher lediglich Nachricht N2 eingetroffen ist. Trifft nun die optionale Nachricht N1 ein, so kann diese getrost verworfen (Cancel) und lediglich der Folgezustand neu gesetzt werden.

Bleibt zum Schluss die Frage offen, was mit den zurückgehaltenen Nachrichten passieren soll. Sie müssen immer dann neu betrachtet werden (und zwar alle gepufferten Nachrichten), sobald eine neu eingetroffene Nachricht vom Prozess durch einen Forward-Eintrag behandelt wurde. Dadurch können sich auch die Voraussetzungen für andere Nachrichten geändert haben, so dass sie jetzt ebenfalls weitergeleitet werden können. Sind die Nachrichten zudem innerhalb der Warteschlange nach ihrer Reihenfolge aufsteigend sortiert, steigt die Wahrscheinlichkeit einer fortlaufenden Abarbeitung bzw. müssen die weiter hinten stehenden Nachrichten gegebenenfalls gar nicht mehr betrachtet werden, wenn schon die erste Nachricht der Warteschlange nicht weiter verarbeitet werden kann.

Abbildung 125 zeigt ein dazu passendes Prozessmodell.

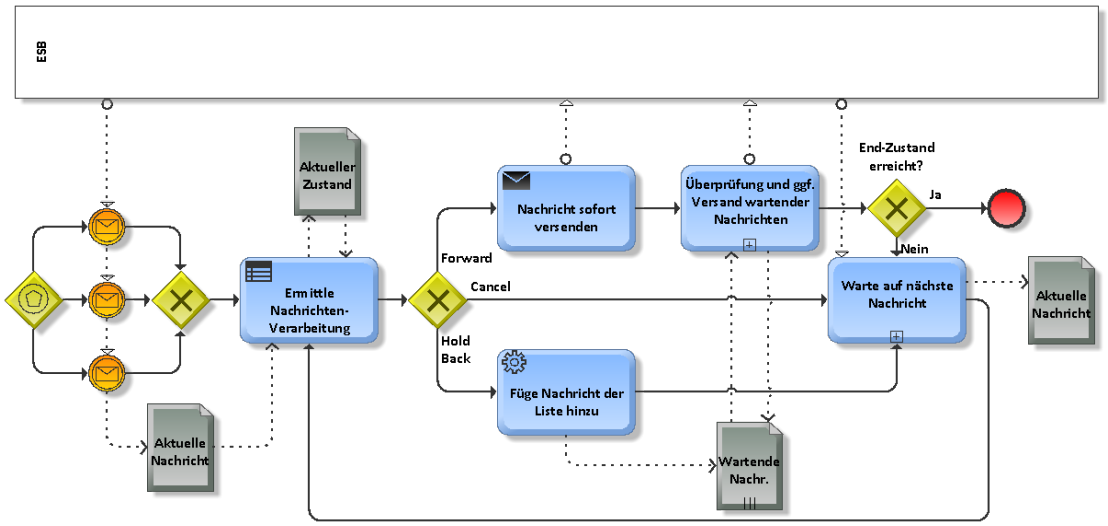



***Abbildung 125: Steuerung komplexer Nachrichtensequenzen als BPMN-Modell***

Da nicht bekannt ist, welche der möglichen Nachrichten als Erste eintrifft, muss der Prozess über ein instanziierendes exklusives ereignisbasiertes Gateway gestartet werden. In obigen Beispiel wird zur Vereinfachung von lediglich drei möglichen Kandidaten ausgegangen. Nach dessen Eintreffen sorgt das Regelwerk für die Ermittlung der weiteren Verarbeitung basierend auf den aktuellen Zustand der Kommunikation. Dabei dient das Datenobjekt *Aktueller Zustand* sowohl als Ein- als auch als Ausgabeparameter der Regel. Das der Geschäftsregelaufgabe folgende Gateway behandelt die vom Regelwerk ermittelten Aktionen *Forward*, *Hold-Back* und *Cancel*. Bei *Hold Back* wird die aktuell in Arbeit befindliche Nachricht einer Warteliste hinzugefügt. Anschließend wird im Unterprozess *Warte auf nächste Nachricht* auf weitere Nachrichten gewartet. Das Warten auf die unterschiedlichen Nachrichtentypen wurde dabei zur Wahrung der Übersicht in einem Unterprozess gekapselt.

Im Falle von *Cancel* kann die Nachricht ignoriert und sofort in den Wartezustand gewechselt werden. Bei *Forward* hingegen kann die aktuelle Nachricht unmittelbar an den Empfänger weitergeleitet werden. Da aufgrund der soeben ausgeführten Aktion nun die Möglichkeit besteht, dass auch andere, noch wartende Nachrichten verschickt werden könnten, überprüft der Unterprozess *Überprüfung und ggf. Versand wartender Nachrichten* diesen Sachverhalt und führt je nach Konstellation entsprechende Aktionen aus. Auch hier wird wieder auf das eingangs verwendete Regelwerk zurückgegriffen. Der Unterprozess beendet sich genau dann, wenn eine Überprüfung der Warteliste keinen weiteren Versandkandidaten mehr identifizieren konnte. Nach Beendigung des Unterprozesses ermittelt das darauffolgende Gateway das Erreichen des

Endzustands. Nur wenn er nicht erreicht wurde, folgt auch für diesen Zweig der Wartezustand. Andernfalls kann der gesamte Prozess beendet werden.

Nach Ausführung des Unterprozesses *Warte auf nächste Nachricht* wird zur Regelaufgabe zurück verzweigt, um aufgrund der neu eingegangenen Informationen die weitere Verarbeitung zu ermitteln, so dass der erläuterte Bearbeitungs-Kreislauf von Neuem beginnt.

Natürlich sind diesem Verfahren Grenzen gesetzt: bei n zu behandelnden Nachrichten müssten 2^n Zustandswechsel betrachtet werden. Allerdings können in der Realität eine Vielzahl von Übergängen ausgeschlossen werden, so dass sich die Anzahl realistischer Übergänge deutlich reduziert. Dennoch zeigt diese Vorgehensweise, wie Nachrichtenbehandlungslogik mittels Regeln umgesetzt werden kann.

Ein letztes Beispiel verdeutlicht den Einsatz von Regeln bei der Validierung von Nachrichten: obwohl Nachrichten syntaktisch einwandfrei an den systemzentrischen Prozess weitergeleitet wurden, können sie dennoch semantisch falsch sein. Insbesondere wenn es um Feldkombinationen geht, können Regeln automatisch fehlerhaft erscheinende Nachrichten aussortieren und an einen Sachbearbeiter zur Kontrolle und Korrektur weiterleiten. Klassische Beispiele für diese Art von Problemen sind sicherlich Fehler bei der Währungsumrechnung. Wie leicht wird statt einer Division mit einem Umrechnungsfaktor eine Multiplikation vorgenommen. Normalerweise bliebe eine solche Nachricht entweder in der Middleware stecken, da die empfangende Applikation die Annahme aufgrund lokal implementierter Validierungslogik verweigert oder sie erreicht das Zielsystem und richtet dort entsprechenden Schaden an, der oftmals nur mühsam wieder korrigiert werden kann. Verbleibt die Nachricht hingegen in der Middleware, so ist die Korrektur des Nachrichteninhalts oftmals schwer möglich, da in Middlewareprodukten aufgrund von Sicherheitsvorkehrungen nicht einfach Werte verändert werden dürfen. Die automatische Aussortierung durch den Einsatz von Validierungsregeln und das Vorsehen einer expliziten Korrektur löst das Problem relativ einfach. Selbst wenn dann noch immer Nachrichten aufgrund nicht berücksichtigter Feldkonstellationen auf Fehler laufen, lässt sich durch Anpassung bzw. Ergänzung des Regelwerks und Neustart der fehlerhaften Nachricht zur Laufzeit eine Lösung finden. Nach der Regelkorrektur wird die Nachricht den Geschäftsregeln erneut

zur Validierung vorgelegt, mit der gewünschten Aussortierung und einer möglichen Korrektur als Folge. Mit jedem weiteren Fehlerfall entwickelt sich das Regelwerk weiter, wird perfekter, und übernimmt in zunehmender Weise vollautomatisch Aufgaben, die zuvor durch Fehler angezeigt wurden und umfangreiche Nacharbeiten erforderten. In der letzten Ausbaustufe können Regeln in einfachen Fällen auch die Korrektur automatisch übernehmen, sofern sie einem immer wiederkehrendem Muster folgen. Dadurch wird erneut der Automatisierungsgrad und damit die Effizienz im Unternehmen gesteigert.

Abbildung 126 zeigt auch für diesen Fall ein passendes Prozessmodell.

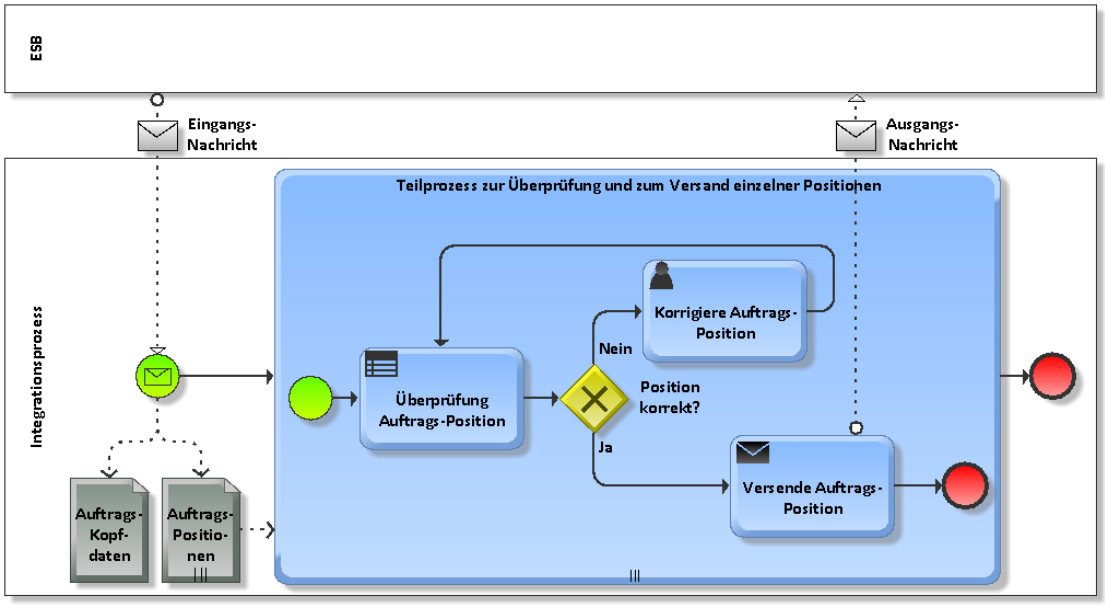



***Abbildung 126: Regelbasierte Feldüberprüfung in systemzentrischen Prozessen***

Die Eingangsnachricht enthält in diesem Beispiel einen Auftrag bestehend aus einem Auftragskopf und mehreren Auftragspositionen. Diese beiden Nachrichtenbestandteile werden zunächst in zwei dafür vorgesehene Datenobjekte aufgeteilt. Anschließend wird in einem parallel ausgeführten Unterprozess für jede Auftragsposition auf Basis eines Regelwerks eine Überprüfung durchgeführt. Abhängig vom Ausgang dieses Tests wird entweder über eine Benutzeraufgabe eine Korrektur der Position initiiert oder dessen sofortiger Versand angestoßen. Im Falle der Korrektur wird sicherheitshalber eine erneute Verifikation über das Regelwerk durchgeführt, um erneute Fehleingaben ausschließen zu können.

Diese Beispiele zeigen, wie Standard-Integrationsprobleme elegant mit Regelwerken adressiert und gelöst werden können. In heutigen ESB-Implementierungen bietet hingegen jeder Hersteller proprietäre Tools zur Lösung dieser Probleme an. Selbst innerhalb eines Produkts werden für die unterschiedlichen Entscheidungen verschiedene Werkzeuge genutzt, obwohl sie mittels einer Regel-Engine einheitlich adressiert werden könnten und dadurch zudem an Flexibilität gewinnen würden. Die Vorteile von Geschäftsregeln sind allerdings so offensichtlich, dass mit dessen zunehmendem Einsatz auch in Integrationsprodukten zu rechnen ist.

### *5.7.3 Steigerung des Automatisierungsgrades durch Kombination von Geschäftsregeln und analytischen Anwendungen*

Die Automatisierungsgrad von Geschäftsprozessen innerhalb von Verbundanwendungen kann unter Einbeziehung analytischer Daten weiter erhöht werden. Für den Zugriff auf analytische Anwendungen zur Prozessoptimierung bieten sich dabei innerhalb eines Prozessmodells zwei typische Einsatzszenarien an:

- Innerhalb einer Serviceaufgabe zur Vorbereitung von regelbasierten Entscheidungen.
- Innerhalb von Benutzeraufgaben in Form von grafisch aufbereiteten Diagrammen zur interaktiven Entscheidungsfindung.

Abbildung 127 zeigt einmal mehr den bekannten Bestellprozess, diesmal erweitert um eine zusätzliche Serviceaufgabe zur Ermittlung speziell aggregierter Daten aus einem Business Warehouse (BW)-System (die zusätzliche Kollaboration mit dem Data-Warehouse wurde im Modell nicht explizit berücksichtigt).

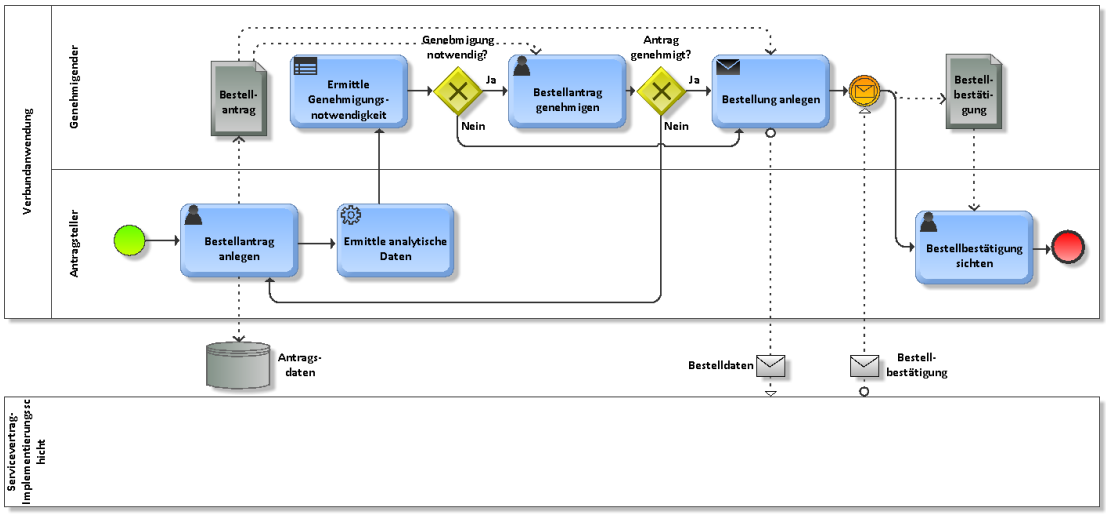



*Abbildung 127: Vereinfachter Bestellprozess ergänzt um analytische Daten*

Unmittelbar nach Eingabe der Daten durch den Antragsteller werden aggregierte Informationen aus einem BW-System gelesen. Dieses stellt beispielsweise vordefinierte Datenanalysen in Form von Web Services zur Verfügung (analytics as a service). Im konkreten Beispiel könnte dies eine Zusammenfassung des Kaufverhaltens des Antragstellers in der Vergangenheit zusammen mit Zahlen über die Zuverlässigkeit seiner Zahlungen sein. Diese Informationen fließen in die Entscheidungsfindung über ein Regelwerk ein. Der Vorteil dieses Ansatzes liegt in der Kombination von transaktionalen und analytischen Daten zur verbesserten Entscheidungsfindung. Die im BW-System hinterlegten, aus unterschiedlichsten Systemen zusammengetragenen und in Beziehung gesetzten Daten ermöglichen eine wesentlich fundiertere und zutreffendere Auswertung und damit qualitativ höherwertige Entscheidungen. Die Bereitstellung eines solchen Dienstes durch das BW-System in Form von Web Services und das Zusammenspiel mit Geschäftsregeln erlaubt eine weitere Automatisierung und Flexibilisierung von Geschäftsprozessen. Anstelle von Sacharbeitern, die mühsam händisch analytische Diagramme und Tabellen auswerten müssen, kann dies nun vollautomatisch durch die Kombination Analysedaten/Regelwerk übernommen werden. Natürlich können nicht alle Entscheidungen automatisiert werden. Von daher bleibt die interaktive Auswertung von Daten durch den Sachbearbeiter weiterhin ein wichtiger Bestandteil von Prozessen. Der Unterschied liegt jetzt allerdings darin, dass der Experte nur noch in deutlich weniger Fällen konsultiert werden muss, da der Großteil der Entscheidungen maschinell abgewickelt wird.

Dies könnte, bezogen auf obiges Prozessmodell, in Form von grafisch aufbereiteten Diagrammen im *Bestellung genehmigen*-Schritt erfolgen: neben den während des Prozessablaufs gesammelten Daten werden ergänzend Informationen aus einem BW-System in die Entscheidungsoberfläche des Sachbearbeiters eingearbeitet. Er bekommt also kontextabhängig die zur Entscheidungsfindung relevanten Daten präsentiert, ohne dass dieser selbst das BW-System aufrufen und Informationen recherchieren muss. Dieses Szenario reflektiert damit die eingangs erwähnte zweite Einsatzmöglichkeit von Auswertungsdaten in Prozessen.

Interessant ist nun zu beobachten, wie nach der Einführung einer derartigen Lösung eine weitere Prozessoptimierung erzielt werden kann. Schwachpunkte sind nach wie vor interaktive Benutzeraufgaben. Von daher bietet sich an, die Arbeitsweise des Sachbearbeiters bei seiner manuellen Entscheidungsfindung zu untersuchen. In der Regel stellt sich dabei heraus, dass er immer wieder dieselben Reports, Transaktionen usw. aufruft, um zu einer Entscheidung zu kommen. Lässt man sich seine Vorgehensweise von dem Experten erklären, so werden immer wieder Formulierungen wie „*Wenn das Feld abc den Wert xyz überschreitet, rufe ich zur Sicherheit noch einen weiteren Report auf, aus dem ich dann den Wert uvw zu meiner Entscheidungsfindung heranziehe. Befindet sich uvw im Bereich aaa bis bbb, so bin ich auf der sicheren Seite und genehmige. Andernfalls lehne ich ab.*“ verwendet. Es ist genau dieses für Unternehmen unschätzbare und über einen langen Zeitraum erworbene Wissen der Mitarbeiter, das sukzessive in Regeln überführt und dadurch für das Unternehmen konserviert werden kann. Gleichzeitig trägt dieses Vorgehen zu einer weiteren Automatisierung bei und bedeutet für den Mitarbeiter, dass er von lästigen Routinearbeiten befreit wird und somit mehr Zeit für die wirklich schwierigen Ausnahmesituationen verbleibt.

Ziel dieses Vorgehens ist also, Wissen und Erfahrungen sukzessive in Regeln zu überführen und für das Unternehmen nutzbar zu machen. Unternehmen und Mitarbeiter profitieren letztendlich in mehrfacher Hinsicht davon:

- der Automatisierungsgrad steigt
- dem Mitarbeiter werden zeitaufwändige und ungeliebte Routinearbeiten abgenommen, was seine Motivation steigen lässt, da er sich intensiver um wirklich

anspruchsvollere und damit interessantere Aufgaben kümmern kann, was wiederum der Qualität zu Gute kommt
- Risikominderung für das Unternehmen durch Ausfall bzw. Weggang des Mitarbeiters
- das extrahierte Wissen der Entscheidungsfindung wird als Blaupause/“Best Practice“ auch für andere Anwendungsfälle herangezogen

Zusammengefasst offenbart die Kombination der drei Bausteine Prozesse, Regeln und die Einbeziehung analytischer Daten ein enormes Optimierungs- und Automatisierungspotenzial. Dabei werden Unternehmen mittlerweile durch geeignete Werkzeuge und Frameworks unterschiedlicher Hersteller optimal unterstützt, so dass sie letztendlich schneller, gezielter und präziser auf sich ändernde Marktbedingungen reagieren können. Sie können dadurch entweder ihre führende Marktposition festigen oder sie sogar noch weiter ausbauen.

## 5.8 Verbundanwendungen und unvorhersehbare Prozessabläufe

Im Laufe dieser Arbeit wurden eine Vielzahl von Prozessmodellen diskutiert. Ihnen allen gemein war eine stets fest vorgegebene Sequenzfolge. Zwar wurde immer Wert auf ein Höchstmaß an Flexibilität gelegt, dennoch lässt sich eine gewisse Statik nicht leugnen. Dies wird auch in der Literatur immer wieder kritisiert. Pucher 2010 bringt es in seinem Blog wie folgt auf den Punkt: „..., dass ich seit zehn Jahren lautstark bis provokativ gegen die Idee protestiere, dass man Unternehmen mittels BPM perfekt strukturieren und damit verbessern kann. Sie werden damit zu Tode organisiert.“ Im Folgenden geht er auf die Forschungen im Bereich der adaptiven Prozesse ein und wie im Laufe der Jahre das von ihm patentierte Verfahren der User-Trained Agents (UTA - benutzertrainiertes Lernen) entstand. Die wesentliche Idee dabei ist, dass Agenten zu jedem Zeitpunkt Benutzeraktionen überwachen und mitprotokollieren. Die durch diese Überwachung gemachten Erfahrungen werden anschließend dazu verwendet, dem Benutzer Vorschläge für den weiteren Ablauf zu machen. Wird der Vorschlag angenommen, stärkt das das Vertrauen in das gesammelte Wissen. Andernfalls sinkt es wieder und der Agent versucht, neue, bessere Muster in den gesammelten Daten zu finden, damit die Erfolgsquote beim nächsten Vorschlag wieder erhöht wird. Im Prinzip ist dieses Vorgehen eine Automatisierung des im vergangenen Ab-

schnitt erläuterten Interviews zur Ermittlung von Entscheidungsregeln. Allerdings hat Pucher nicht das Ziel, die menschlichen Interaktionen völlig zu eliminieren, sondern den Anwender bei der Fortsetzung des Prozesses behilflich zu sein, ohne diesen starr vorab definieren zu müssen. In seine Arbeit fließen auch Ideen des Process Mining (siehe auch Web-Seite zu Process Mining von van der Aalst 2010 mit weiteren Forschungsarbeiten zu diesem Thema) ein, bei dem aus den Prozess-Event-Logs die Basis für Änderungsvorschläge gebildet wird. Doch wie können nun völlig unvorhersehbare Prozesse wie eine medizinische Behandlung in einem Krankenhaus, die Abwicklung von Gerichtsverfahren, die Erstellung eines kundenindividuellen Angebots oder die Begutachtung eines Versicherungsschadens (Allweyer 2010b) IT-technisch unterstützt werden? Im Rahmen dieser Arbeit soll daher im Folgenden betrachtet werden, was die BPMN für solche Szenarien vorsieht und wie durch eine Kombination von strukturierten, unstrukturierten und Standard-Prozessen das Problem adressiert werden kann. Für weitere Informationen zum Thema „unvorhersehbare Prozesse“ sei auf Swenson 2010 verwiesen.

Standardmäßig sieht die BPMN-Spezifikation zur Abbildung von nicht fest vorgegebenen Ausführungssequenzen von Aufgaben den sogenannten Ad-Hoc-Teilprozess vor. Abbildung 128 zeigt einen solchen Teilprozess mit mehreren Aufgaben.

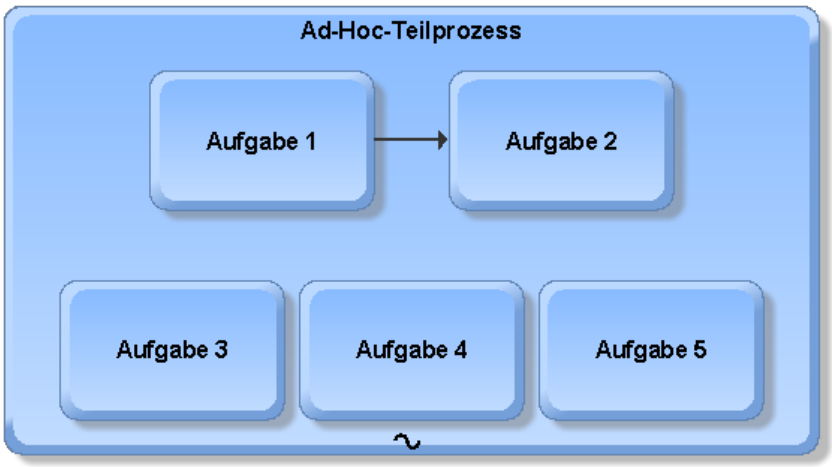



*Abbildung 128: Ad-Hoc-Teilprozess*

Semantisch legt die Spezifikation die Behandlung eines solchen Teilprozesses wie folgt fest: zunächst werden sämtliche Aufgaben ohne eingehenden Sequenzfluss aktiviert (in obigen Beispiel also die Aufgaben 1, 3, 4 und 5). Eine der aktivierten Aufga-

ben wird zur Ausführung selektiert. Ist das dem Teilprozess zugeordnete Attribut `ordering` zudem auf Parallelverarbeitung gesetzt, dürfen auch mehrere aktivierte Aufgaben gleichzeitig ausgeführt werden, was im Falle einer sequenziellen Abarbeitung natürlich ausgeschlossen ist. Nach der Beendigung einer Aufgabe wird die Beendigungsbedingung (`completionCondition`) für den Teilprozess ausgewertet: ist sie nicht erfüllt, wird die Menge der aktivierten Aufgaben neu ermittelt und gegebenenfalls zur Ausführung gebracht. Ist die Bedingung hingegen erfüllt, wird der Ad-Hoc-Teilprozess beendet, ohne dass weitere Aufgaben neu aktiviert werden. Bei einer Parallelverarbeitung kann es allerdings passieren, dass noch Aufgaben bearbeitet werden, obwohl die Beendigungsbedingung mittlerweile erfüllt wird. Dann ist die Behandlung dieser noch laufenden Aufgaben vom Wert eines weiteren Attributs des Teilprozesses abhängig: ist nämlich der Wert des Attributs `cancelRemainingInstances` auf den booleschen Wert `true` gesetzt, werden sämtliche noch aktiven Aufgaben unterbrochen und der Sequenzfluss unmittelbar mit der dem Teilprozess folgenden Aufgabe fortgesetzt. Im Falle von `false` wird hingegen auf die Beendigung aller noch in Bearbeitung befindlicher Aufgaben gewartet.

Durch den Ad-Hoc-Teilprozess wird also die Statik der fest vorgegebenen Ausführungssequenzen aufgebrochen. Allerdings müssen auch in diesem Fall sämtliche Aufgaben zur Designzeit bekannt sein. Was ist aber, wenn während der Laufzeit festgestellt wird, dass eine neue Aufgabe benötigt wird, die so nicht vorhersehbar war? Bei einem Krankenhausaufenthalt wird dies besonders deutlich. Wie Allweyer 2010b richtig feststellt, wird ein Arzt abhängig von Diagnose und Krankheitsverlauf verschiedene Untersuchungen und Behandlungen veranlassen. Allein mit einem Ad-Hoc-Teilprozess lässt sich dieses Verhalten nicht abbilden. Hier kommt die sogenannte *Fallakte* ins Spiel, die sämtliche Informationen wie Analysen, Berichte, Bilder, Diagramme usw. aufnimmt und die kollaborativ von den Prozessbeteiligten bearbeitet wird. Doch wie kann die Fallakte mit strukturierten Prozessen zusammenarbeiten um eine ganzheitliche Prozessabdeckung zu ermöglichen? Schließlich gibt es auch bei Krankenhausaufenthalten Prozesse, die durchaus strukturiert ablaufen und auch so von der IT unterstützt werden sollten. Ein konkretes Beispiel veranschaulicht Abbildung 129 auf der übernächsten Seite, in dem das Zusammenspiel von strukturierten, unstrukturierten und Standard-Prozessen gezeigt wird.

Der Prozess beginnt mit der Patientenaufnahme: im Schritt *Erstaufnahme und Zuweisung Arzt* werden die üblichen Patientendaten aufgenommen und ein diensthabender Arzt ausgewählt, der während des Krankenhausaufenthalts für diesen Patient verantwortlich ist. Der Arzt hat nach einer eingehenden Untersuchung zu unterscheiden, ob weitere Untersuchungen notwendig sind. Ist das nicht der Fall, kann der Patient unmittelbar die Klinik verlassen. Andernfalls wählt der Arzt aufgrund des aktuellen Erkenntnisstands die Fachärzte aus, von denen er weitere Erkenntnisse für eine Diagnose erwarten kann und startet die Kollaboration. An dieser Stelle wird der fest vorgegebene Prozessablauf verlassen. Zwar erscheint im BPMN-Diagramm als Aktion im *Kollaboration*-Pool eine Ad-Hoc-Aktivität. Doch diese ist dort lediglich als Stellvertreter für eine eigenständige Software-Lösung platziert, die sich auf die Zusammenarbeit zwischen den Ärzten konzentriert. Derartige Lösungen sind auch als Groupware bekannt und unterstützen die Kollaboration über räumliche und zeitliche Distanz hinweg. Neuere Lösungen bringen neben Menschen und Informationen auch Arbeitsmethoden in Form von dedizierten Werkzeugen zusammen. Zu solchen Methoden zählen beispielsweise Rankings, Pro- und Kontra-Tabellen, Schnellumfragen, SWOT-Analysen (Strength, Weaknesses, Opportunities, Threads), Kosten-Nutzen-Analysen und dergleichen. Die Teilnehmer können zu jeder bereitgestellten Information Diskussionen beginnen und kommentieren. Letztendlich wird das Ziel verfolgt, die Nachvollziehbarkeit der Entscheidungsfindung gewährleisten zu können. Da diese auch zu späteren Zeitpunkten einsehbar sein muss, ist eine Archivierungsfunktion unverzichtbar. Ein Beispiel für ein derartiges Produkt mit den genannten Eigenschaften ist SAP StreamWork. Weitere Details dazu finden sich unter SAP 2011.

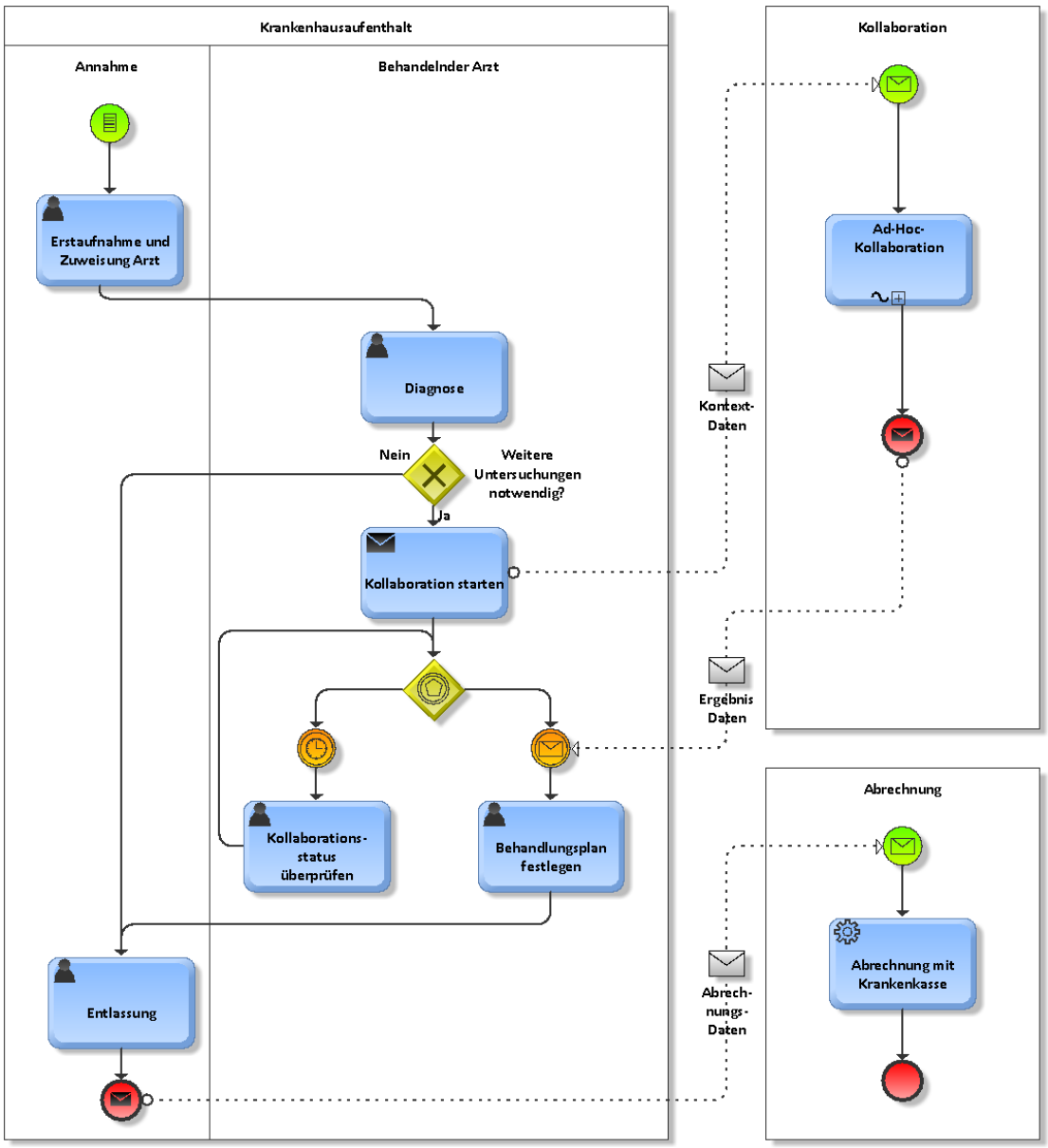



***Abbildung 129: Zusammenspiel zwischen strukturierten, unstrukturierten und Standard-Prozessen als BPMN-Modell***

In dem konkreten Beispiel wird das Zusammenspiel zwischen Geschäftsprozess und Kollaborationssoftware über das Anlegen einer neuen Kollaboration durch den Geschäftsprozess, der Auswahl der an dieser Kollaboration beteiligten Ärzte sowie der Übergabe der bereits erfassten Patientendaten gewährleistet. Die Sende-Aufgabe *Kollaboration starten* ist im Prozessmodell für den Start verantwortlich. Anschließend begibt sich der strukturierte Prozess in eine Warteposition. Die Kollaboration verläuft nun völlig autark ohne weitere Interaktion mit dem behandelnden Arzt. Der Hauptprozess wartet am ereignisbasierten Gateway entweder auf den Ablauf eines Timers oder

auf das Eintreffen der Ergebnisdaten aus der Kollaborationssoftware. Der Timer ermöglicht die Überwachung der weiteren Behandlung und ein Einschreiten, falls die Untersuchungen nicht wie gewünscht durchgeführt werden. Bei kritischen Fällen können die beteiligten Fachärzte auch auf die besondere Dringlichkeit hingewiesen und dadurch eine beschleunigte Behandlung erreicht werden.

Die Kollaborationssoftware ermöglicht das Zusammenfassen sämtlicher Untersuchungsergebnisse in einer Fallakte. Jeder Beteiligte hat die Möglichkeit, sich Einsicht in die bisher vorliegenden Ergebnisse zu verschaffen. Sollte sich dabei herausstellen, dass zu den bereits verordneten Untersuchungen weitere benötigt werden, so fügen die Fachärzte diese, ggf. nach Rücksprache mit dem betreuenden Arzt, hinzu. An dieser Stelle wird der besonderen Dynamik von unstrukturierten Prozessen Rechnung getragen. Situationsabhängig entscheiden die Prozessbeteiligten über den weiteren Verlauf. Unter Umständen werden Personen und Abteilungen hinzugezogen, die so in den Prozessen noch nicht einbezogen wurden und deren Beteiligung in einem strukturierten Prozess nicht berücksichtigt würde. Kollaborationssoftware erlaubt genau diesen Flexibilitätsgrad.

Wurde die letzte Untersuchung abgeschlossen, kann der Hauptprozess über eine passende Nachricht darüber informiert und aus dem Wartezustand befreit werden. Der koordinierende Arzt kann aus seiner Anwendungsoberfläche heraus auf die Ergebnisse der Kollaboration zurückgreifen und einen passenden Behandlungsplan festlegen. Ab jetzt läuft der Prozess wieder strukturiert ab: vor Verlassen des Krankenhauses verabschiedet er sich offiziell bei der Patientenannahme und bekommt ggf. Rezepte, Überweisungen oder andere medizinische Unterlagen ausgehändigt. Beim Ende-Ereignis wird schließlich noch ein standardisierter Prozess angestoßen: die Abrechnung mit der Krankenkasse des Patienten kann vollautomatisch durch Standardsoftware abgewickelt werden. Der Hauptprozess triggert lediglich deren Ausführung.

Dieses Beispiel zeigt das Zusammenspiel der Prozessarten strukturiert, unstrukturiert und standardisiert, wobei der mit BPMN modellierte Hauptprozess einmal mehr die Führung übernommen und die Aktivitäten koordiniert hat.

Das beschriebene Szenario wurde bei der SAP in einem Prototypen implementiert. Dabei kamen SAP NetWeaver BPM als Umgebung für den strukturierten Prozess und SAP StreamWork als Kollaborationsplattform zum Einsatz. Als Backend fungierte

SAP ERP zur Abwicklung des Standardprozesses. Die folgenden Abbildungen zeigen das Ergebnis ergänzt um einige Erläuterungen zur Implementierung. Damit konnte unter Beweis gestellt werden, dass anspruchsvolle Aufgabenstellungen auch unter Berücksichtigung unvorhersehbarer Prozessstrecken mit heutigen Mitteln zufriedenstellend gelöst werden können.

Abbildung 130 zeigt den Bildschirm zur Aufnahme des Patienten.

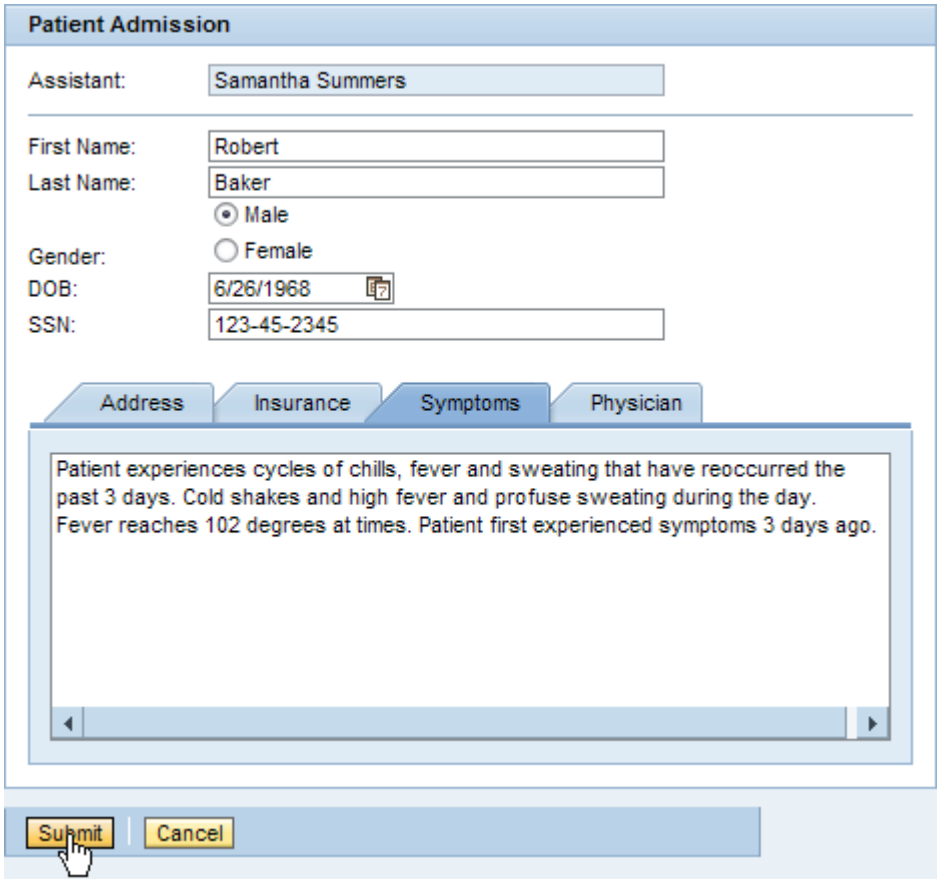



***Abbildung 130: Formular zur Patientenaufnahme***

Der Bildschirm zeigt eine typische Erfassungsmaske mit verschiedenen Reitern zur übersichtlicheren und gleichzeitig geführten Eingabe der Patienteninformationen wie Adresse, Symptome, Versicherungsdaten sowie den diensthabenden Arzt zur Erstuntersuchung (Lasche *Physician*). Nach Beendigung der Eingaben wird dem ausgewählten Arzt eine Information zugeschickt, so dass ihm die Daten bei Erscheinen des Patienten vorliegen.

Bei der Erstuntersuchung fallen bereits erste Ergebnisse an, die auch in der späteren Zusammenarbeit mit den Fachärzten von Bedeutung sind. Von daher muss ein Upload für eingescannte Unterlagen oder erstellte Bilder (Kardiogramme, Ultraschallbilder usw.) möglich sein. Abbildung 131 zeigt die Benutzeroberfläche des behandelnden Arztes, die noch Bestandteil des strukturierten Prozesses ist.

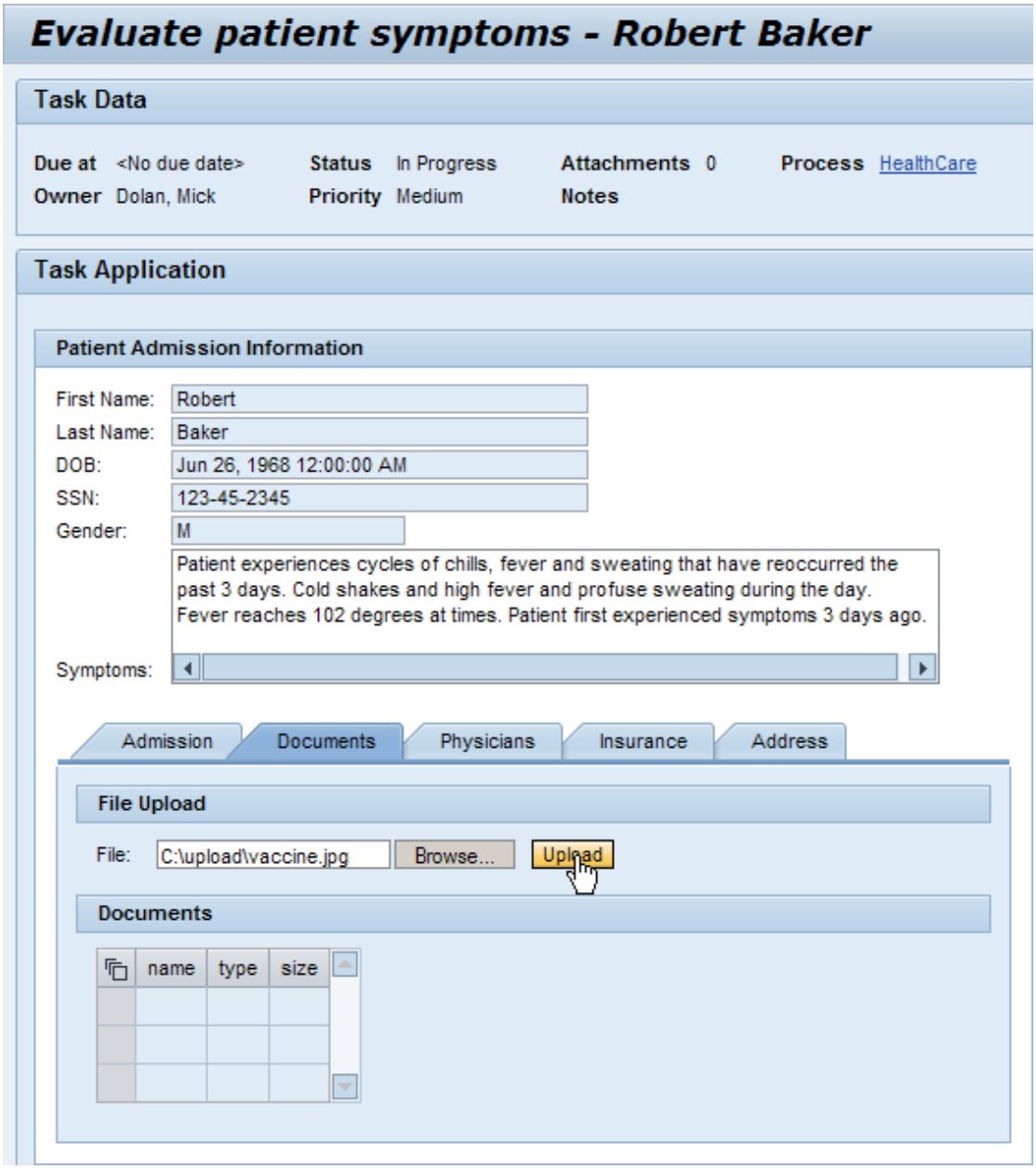



*Abbildung 131: Formular zur Eingabe der Erstuntersuchung*

In diesem Beispiel wird soeben der Impfausweis des Patienten hochgeladen. Ansonsten ist die Oberfläche mit dem bereits gezeigten Aufnahmebildschirm nahezu

identisch. Lediglich die *Physicians*-Lasche ist um eine wichtige Funktionalität erweitert worden: der Initiierung der Kollaboration. Auf ihr werden die Fachärzte ausgewählt, die weitere Daten wie Laborwerte, EKG, Röntgenaufnahmen oder Ultraschalluntersuchungen zu ergänzen haben. Abbildung 132 zeigt die vom Betreuungsarzt Mick Dolan ausgewählten Spezialisten.

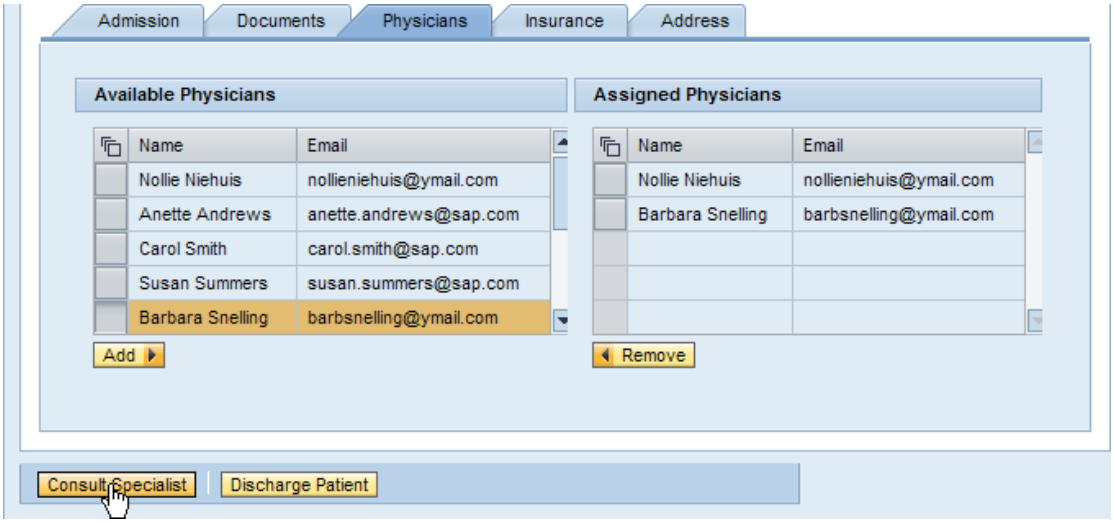



***Abbildung 132: Auswahl der Fachärzte für die Kollaboration***

Durch Betätigung des *Consult Specialist*-Druckknopfs wird die Kollaboration in SAP StreamWork angelegt. SAP StreamWork ist mit einem REST-API ausgestattet, das nicht nur den Start einer Kollaboration erlaubt, sondern das auch den Upload von Dokumenten, den erfassten Daten und den beteiligten Personen ermöglicht. Aus dem Programm der Benutzeroberfläche heraus werden also all diese Informationen an SAP StreamWork übergeben. Nach Anlage der Kollaboration kann sich auch dessen Initiator (Mick Dolan) sofort mit der Plattform verbinden. Ein Dialog macht den Arzt über die erfolgreiche Instanziierung und der Möglichkeit, sich mit dem Workspace zu verbinden, aufmerksam (Abbildung 133).

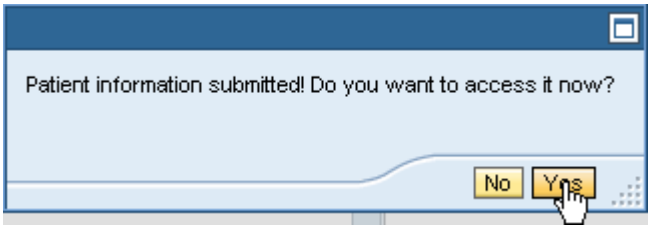



***Abbildung 133: Bestätigung der erfolgreichen Instanziierung der Kollaboration***

Nachdem sich der Arzt mit der Kollaborationsplattform verbunden hat, erscheint die in Abbildung 134 dargestellte Umgebung von SAP StreamWork. Gut zu erkennen ist die Zweiteilung des Bildschirms in eine linke Hälfte, die die jeweiligen Informationen wie in diesem Beispiel die Daten des Patienten und den Impfplan zeigt, und in eine rechte Hälfte, die es jedem Beteiligten ermöglicht, zu den Informationen auf der linken Seite einen Kommentar abzugeben. Es sind gerade die aus diesen Diskussionen resultierenden Ergebnisse, die in den meisten Fällen nicht oder nur unzureichend dokumentiert sind. In SAP StreamWork stellen diese Diskussionsfäden einen fundamentalen Bestandteil der Lösung selbst dar und sind dabei gleichzeitig den jeweiligen Informationen (linke Hälfte des Bildschirms) eindeutig zugeordnet.

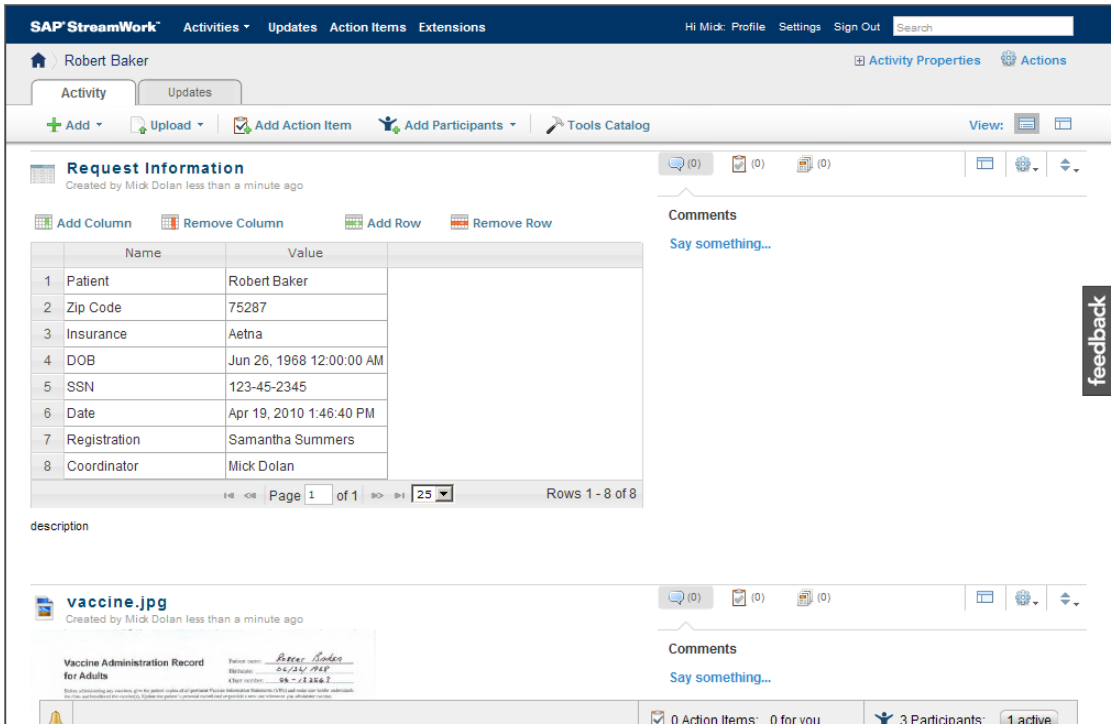



***Abbildung 134: Kollaborationsumgebung SAP StreamWork***

In der unteren rechten Ecke des Bildschirms sind die Kollaborationsbeteiligten erkennbar. Derzeit arbeiten in dieser Kollaboration drei Teilnehmer zusammen, wovon einer aktiv ist. Bei Klick auf den Text *3 Participants* erscheinen die vom behandelnden Arzt ausgewählten Spezialisten, die folglich ebenfalls erfolgreich an SAP StreamWork übergeben wurden (Abbildung 135).

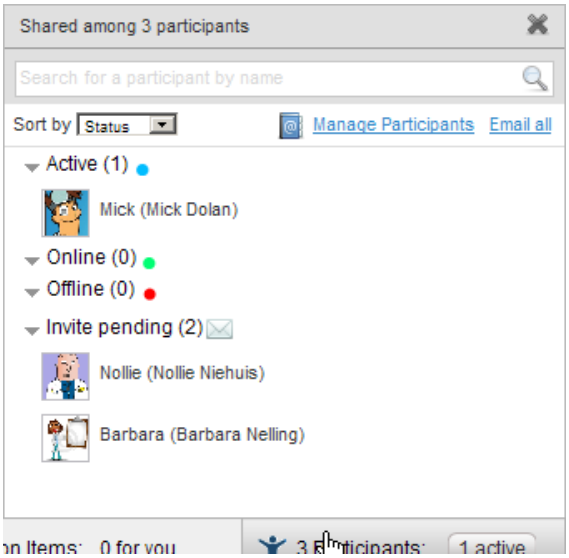



***Abbildung 135: Auflistung der an der Kollaboration beteiligten Ärzte***

Die Spezialisten können nun bei Eintreffen des Patienten auf die bereitgestellten Daten in der Fallakte zurückgreifen und sich ein umfassendes Bild vom Patienten machen. Sollte sich bei den laufenden Untersuchungen die Notwendigkeit weiterer Beteiligungen von Spezialisten herausstellen, so ist dies über den *Add Participants*-Eintrag im Menü jederzeit möglich.

Nach Beendigung der Kollaboration muss der wartende Hauptprozess noch aus seiner Wartesituation befreit werden. Dies erfolgt über einen von SAP StreamWork an SAP NetWeaver BPM gerichteten Web Service-Aufruf. Der Prozess wartet an einem nachrichtenbasierten Zwischenereignis, das in SAP NetWeaver BPM bekanntlich mit einer über Web Service aufrufbaren Schnittstelle verbunden ist. In SAP StreamWork ist diese Funktionalität innerhalb einer mit *Decision* benannten Aktivität verborgen (siehe Abbildung 136).

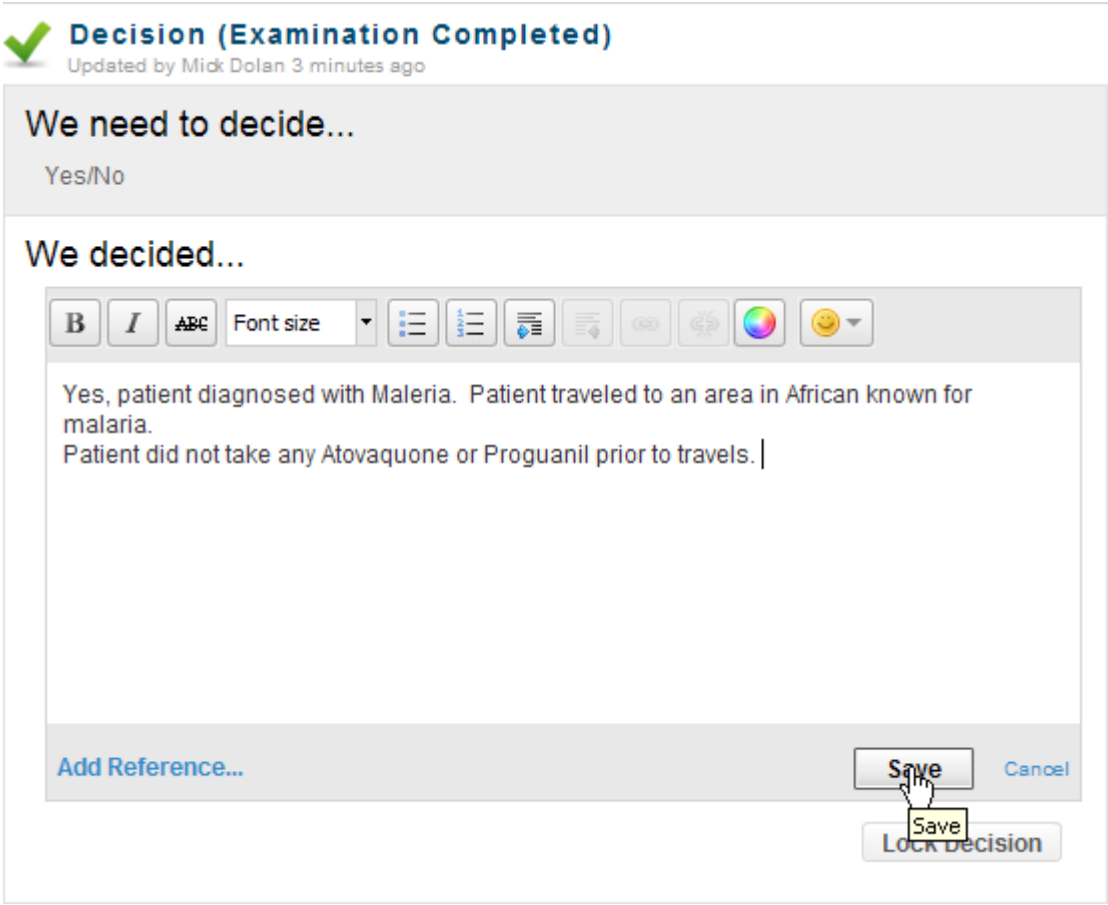



***Abbildung 136: Beendigung der Kollaboration und Reaktivierung des wartenden Hauptprozesses***

Sobald der *Save*-Druckknopf betätigt wird, erfolgt die Kommunikation mit dem wartenden Prozess. Dieser erreicht nun den nächsten Schritt für den koordinierenden Arzt und fordert diesen auf, seine Diagnose zu treffen, den Behandlungsplan festzulegen und die weiteren Schritte zu veranlassen. Abbildung 137 zeigt diesen Schritt im Detail:

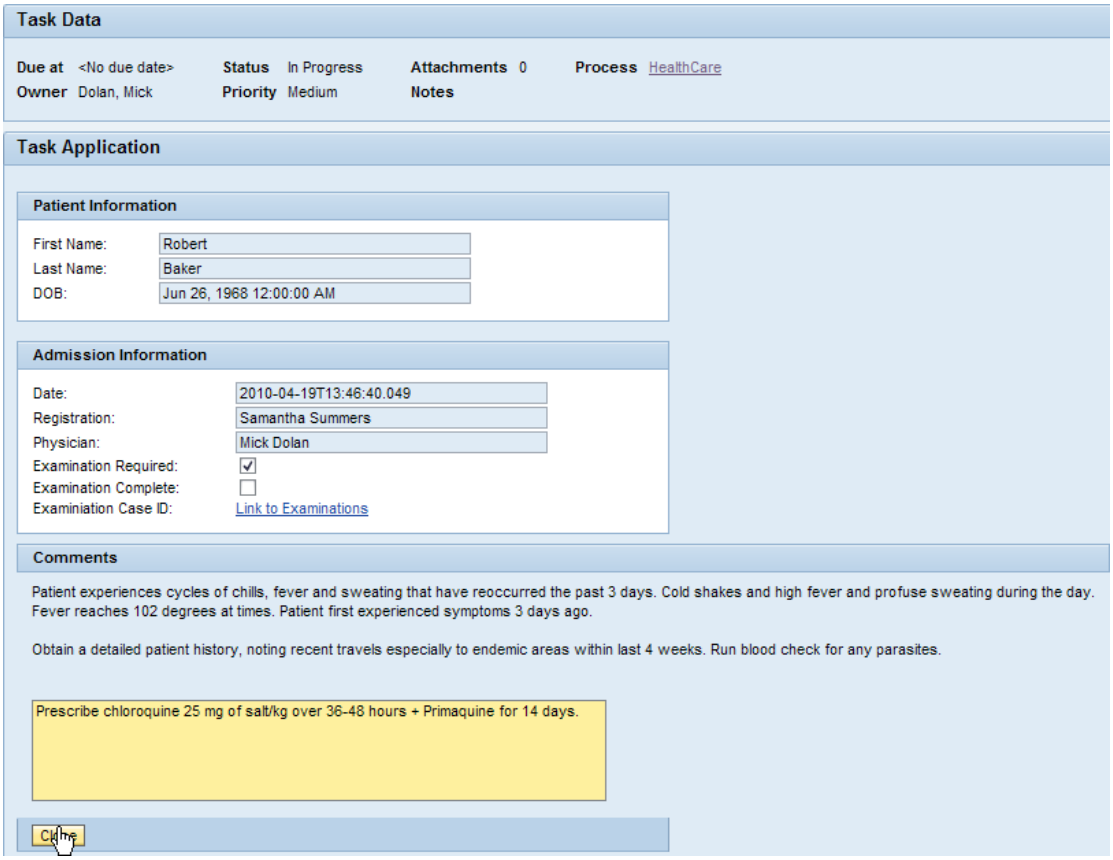



***Abbildung 137: Formular zur Beendigung der Behandlung***

Auch diesmal hat der Arzt erneut die Möglichkeit, über den in der Bildschirmmitte befindlichen Hyperlink *Link to Examinations* zur Fallakte zu navigieren und sich dort nochmals den Verlauf der Untersuchungen zu vergegenwärtigen und nachzuvollziehen, welche Ergebnisse ermittelt wurden, um darauf aufbauend das weitere Vorgehen festzulegen. In diesem Beispiel beschränkt er sich auf das Verschreiben von Medikamenten. Wie dem Prozessmodell zu entnehmen ist, werden diese Informationen schließlich zur Patientenannahme weitergeleitet, wo der Patient sein Rezept entgegennehmen kann und offiziell verabschiedet wird.

# 6 Fazit

Unternehmen stehen vor enormen Herausforderungen. Sie müssen sich im alltäglichen Wettbewerb behaupten und in immer kürzeren Abständen ihre gewinnbringenden Prozesse adaptieren. In dieser Arbeit wurde eine bestimmte Anwendungskategorie, nämlich die der Verbundanwendung bzw. der Composite Application diskutiert, die Unternehmen bei dieser schwierigen Aufgabe unterstützen können. Der wesentliche Kerngedanke ist der einer nachhaltigen Anwendungsarchitektur, die die Unternehmen befähigt, auch bei einer langen Einsatzdauer der Anwendungen deren Flexibilität zu erhalten, die bei den meisten monolithischen Applikationen längst verlorengegangen ist. Eine derartige Architektur kann allerdings nur erreicht werden, wenn sämtliche Komponenten aufeinander abgestimmt sind und bestimmten Prinzipien folgen. Ziel dieser Arbeit war es, diese Prinzipien herauszuarbeiten und konkrete Implementierungen zu diskutieren, die bei der Umsetzung behilflich sind. Dabei wurde eine Composite stets als ein harmonisches Ganzes betrachtet, das sich aus einer Vielzahl unterschiedlichster Komponenten zusammensetzt und das erst im Zusammenspiel seine Fähigkeiten unter Beweis stellt. Der damit einhergehende Flexibilitätsgewinn ist jedoch nur zu erzielen, wenn gleichzeitig sowohl bei den Architekten als auch bei den Entwicklern ein Umdenken stattfindet: Denkmuster, die sich bei der Entwicklung monolithischer Anwendungen entwickelt haben, in der neuen lose-gekoppelten Welt anwenden zu wollen, sind zum Scheitern verurteilt. Diese Unterschiede zu erkennen und geeignete Lösungsmöglichkeiten aufzuzeigen war ebenfalls ein zentrales Ziel dieser Arbeit.

Dazu wurde in Kapitel 1 zunächst erläutert, warum Verbundanwendungen in heutigen Unternehmen nicht fehlen dürfen. Aufgrund ihrer einzigartigen Eigenschaft der Anpassungsfähigkeit und damit der Möglichkeit, schnell und gezielt auf sich ändernde Marktgegebenheiten reagieren zu können, bilden sie den Mittelpunkt einer auf Flexibilität ausgerichteten Unternehmensstrategie. Allerdings muss den Unternehmen bewusst sein, dass sie bereit sein müssen, dafür auch einen Preis zu zahlen: wie im Laufe der Arbeit gezeigt wurde, ist die Entwicklung von Verbundanwendungen anspruchsvoller und erhöht zweifellos die Komplexität. Doch durch die Berücksichtigung einer nachhaltigen Architektur lässt sich die Komplexität beherrschen und zum Nutzen des Unternehmens einsetzen.

In Kapitel 2 wurde daraufhin das Verständnis über Verbundanwendungen durch eine Definition und anschließender Diskussion der typischen Charakteristika geschärft. Mehrere Beispiele unterstrichen die zuvor genannten Eigenschaften. In diesem Zusammenhang wurde auch die Rolle der Business Process Model and Notation (BPMN) für Verbundanwendungen erstmals herausgestellt. Gerade ihr Einsatz als gemeinsame Sprache zwischen der fachlichen Seite und der IT überbrückt den oft zitierten Graben zwischen diesen beiden Parteien. Dabei beschränkt sich der Einsatz von BPMN in Verbundanwendungen nicht nur auf die fachliche Modellierung. In dieser Arbeit wurde ihr Einsatz bewusst auch auf die Modellierung und Ausführung technischer Prozesse erweitert. Sie vermeidet daher die in vielen anderen Arbeiten geschilderte Problematik der Übersetzung von BPMN in ausführbare Modelle wie BPEL.

Nachdem auf diese Weise das Fundament über das Verständnis von Verbundanwendungen gelegt wurde, konnte in Kapitel 3 die Architektur von Composites detailliert besprochen werden. Auffällig dabei war ein konsequenter Top-Down-Ansatz zur Ermittlung der wesentlichen Komponenten, aus denen sich eine Verbundanwendung zusammensetzt, wie z.B. Prozesse, Benutzeroberflächen, Geschäftsobjekte, Ereignisse und Services. Diese sind geeignet in einer Spezifikation zusammenzutragen. Welche Kriterien dabei eine Rolle spielen und wie die Anforderungen in die Spezifikation einfließen, wurde ebenfalls in diesem Kapitel besprochen. In diesem Zusammenhang erwies sich zudem die durchgängige Verwendung eines kanonischen Datenmodells für die Datentypen einer Composite als vorteilhaft: durch dessen Einsatz kann innerhalb einer Verbundanwendung mit aufeinander abgestimmten Typen gearbeitet werden. Transformationen zwischen verschiedenen Datentypen werden dadurch überflüssig.

Ausgehend von einer derart entwickelten Spezifikation setzte sich Kapitel 3.3 anschließend mit den einzelnen Schichten einer Composite Application auseinander. Zur Steigerung der Flexibilität wurde dazu das Konzept der losen Kopplung aus verschiedenen Blickwinkeln betrachtet und der Einsatz für Verbundanwendungen diskutiert. Als Ergebnis konnte eine konsequente Trennung zwischen der *Business Composition* mit Fokus auf benutzerzentrische Prozesse und der *Technical Composition* mit eindeutigem Fokus auf systemzentrische Prozesse empfohlen werden. Dieses Resultat stellt somit einen Kernaspekt der Architektur von Verbundanwendungen dar. Erst diese klar definierte Aufgabenteilung und die Festlegung eines eindeutigen Vertrags zwischen diesen beiden Schichten ermöglicht eine flexible und effiziente Reaktion auf sich

ständig ändernde Marktgegebenheiten. Diese Grundarchitektur bildete für die nachfolgenden Diskussionen um Transaktions- und Fehlerbehandlung, Persistenz, Ereignisverarbeitung usw. die Basis. Dabei wurde auch das Zusammenspiel zwischen Composite Application und der sogenannten Servicevertrag-Implementierungsschicht detailliert betrachtet und gezeigt, wie BPMN durch die Verwendung des nachrichtenbasierten Zwischenereignisses bei der Implementierung eingesetzt werden kann. Obwohl mit der beschriebenen Grundarchitektur bereits ein hohes Maß an Flexibilität gegeben ist, zeigte Kapitel 3.3.2.4, wie diese durch den Einsatz eines ESB's weiter gesteigert werden kann. Die in 3.3.2.4 erarbeitete Architektur stellt somit für Unternehmen die interessanteste Alternative dar, da sie im Hinblick auf die Anpassungsfähigkeit an Änderungen das größte Potenzial besitzt.

In diesem Kontext wurde zudem auf die erweiterte Rolle eines Enterprise Services Repositories (ESR) eingegangen. Stellt sie in den üblichen SOA-Projekten eine Governance-Infrastruktur für *bereitgestellte* Services zur Verfügung, hat sie für Verbundanwendungen auch deren Servicekontrakte zu verwalten. Durch diese Kontrakte werden *die zu implementierenden* Services in Form von betriebswirtschaftlichen Komponenten ausgedrückt, von deren Existenz die Composite ausgeht. Wie diese Funktionalität anschließend in der Servicevertrag-Implementierungsschicht umgesetzt wird, ist für die Composite Application letztendlich unerheblich. Wichtig ist nur, dass zu jeder Composite aufgrund des ESR-Inhalts unmittelbar ersichtlich ist, welche betriebswirtschaftliche Logik konkret implementiert und in welcher Form (Schnittstelle) diese zur Verfügung gestellt werden muss.

Kapitel 4 widmete sich anschließend der konkreten Implementierung einer Verbundanwendung, bei deren Umsetzung den zuvor diskutierten Prinzipien der Grundarchitektur gefolgt wurde. Unter Verwendung des SAP NetWeaver Composition Environments als Modellierungs- und Ablaufumgebung für die mittels BPMN erstellten betriebswirtschaftlichen und technischen Prozesse, konnte ein komplettes Bestellszenario umgesetzt werden, an dem die Anwendung der Prinzipien für Verbundanwendungen exemplarisch demonstriert wurde.

Offensichtlich spielt die Komponentisierung (separation of concerns) bei der Entwicklung von Composites eine entscheidende Rolle. Von daher lag es nahe, zusätzliche Flexibilität durch die Konfiguration von Komponenten mittels Varianten-Komponenten-Diagrammen einzubringen. VCDs stellen auf grafische Art Zusammenhänge

zwischen Komponenten dar und erlauben bei mehreren Alternativen für eine Funktionalität die Auswahl einer passenden Lösung. Dieser Ansatz wurde dem der dynamischen Zuweisung mittels Regelwerken gegenübergestellt und Empfehlungen sowohl für den Einsatz von VCDs als auch von Regelwerken gegeben.

In Kapitel 5 wurden zur Abrundung des Themas weitere Aspekte, die sich aufgrund der Grundarchitektur ergaben, näher untersucht. Fragestellungen wie...

- Welches Locking-Verhalten müssen die angeschlossenen Systeme unterstützen?
- Warum wird Idempotenz für Service-Aufrufe benötigt und wie kann dieses Problem gelöst werden?
- Wie ist auf Ereignisse und Ausnahmesituationen geeignet zu reagieren?
- Welchen Einfluss hat die Architektur auf Benutzeroberflächen?

...waren Gegenstand der Diskussion. Auch in diesen Fällen sind jeweils verschiedenste Lösungsmöglichkeiten aufgeführt sowie Vor- und Nachteile betrachtet worden. Als Ergebnis wurden typische BPMN-Pattern für verschiedenste Problemstellungen innerhalb einer Composite (Composition Pattern), aber auch innerhalb der Servicevertrag-Implementierungsschicht (Systemzentrische Pattern), entwickelt, durch deren Einsatz die Implementierung komplexer Szenarien effizienter gestaltet werden kann.

Aufgrund der Ergebnisse der zuvor betrachteten systemzentrischen Prozesspattern wurde zur Abrundung dieses Themas eine Erweiterung der BPMN vorgeschlagen, um die Modellierung insbesondere von Integrationsprozessen zu vereinfachen und dabei gleichzeitig die Semantik zu präzisieren. Die wesentliche Idee dabei war eine geeignete Verschmelzung der bei Integrationsentwicklern bekannten, von Hohpe und Woolf eingeführten Notation für Systemintegrationen mit der BPMN. Anhand konkreter Beispiele konnte der Nutzen einer solchen Kombination verdeutlicht werden.

Zum Abschluss des fünften Kapitels wurden schließlich weitere Aspekte zur Steigerung der Flexibilität wie der Einsatz von Regelwerken allein oder im Zusammenspiel mit analytischen Anwendungen sowie die Kollaboration zwischen Standardprozessen, strukturierten Prozessen und unvorhersehbaren Prozessen dargestellt. Die Verfügbarkeit von Werkzeugen wie Kollaborationssoftware, Regel- und Prozessmaschinen sowie analytischen Anwendungen gepaart mit einer nachhaltigen Architektur, so

wie sie in dieser Arbeit erläutert wurde, eröffnet den Unternehmen genau die Freiheitsgrade, die sie zur Entwicklung ihrer unternehmenskritischen Prozesse benötigen, ohne dabei in dieselbe Sackgasse zu geraten, die bei monolithischen Anwendungen schließlich zu einer nahezu nicht mehr wartbaren Software führten und die Innovationen im Kern erstickten. Natürlich ist diese Architektur auch mit Kosten verbunden, die auch stets gegen den zu erwartenden Nutzen abzugleichen sind. Dennoch bliebe als Alternative nur, im Status Quo zu verharren und auf schnelle Prozessänderungen zu verzichten. Dies kann sich ein Unternehmen allerdings aufgrund der fundamentalen Bedeutung eben dieser für das Unternehmen gegenüber den Wettbewerbern differenzierenden Prozesse kaum leisten. Von daher öffnet die in dieser Arbeit entwickelte Architektur einen Weg, die eine Balance zwischen den Vor- und Nachteilen schafft und dabei gleichzeitig in einem vernünftigen wirtschaftlichen Verhältnis steht.

In dieser Arbeit spielte die Flexibilität stets eine zentrale Rolle. Der gesamte Architekturansatz orientierte sich in wesentlichen Teilen an den Bedürfnissen des Managements, die Unternehmensstrategie zeitnah durch die IT in ausführbare Prozesse umzusetzen. Dafür ist die Berücksichtigung von Flexibilität in der Architektur eine wesentliche Notwendigkeit. Eine besondere Art der Flexibilität bietet sich allerdings noch für weitere Forschungsarbeiten an: die der externen Erweiterbarkeit von Verbundanwendungen, ohne allerdings Eingriffe in die ursprünglichen Modelle und den Sourcecode vornehmen zu müssen (sogenannte modifikationsfreie Erweiterungen). Wie können also Composites um neue Felder erweitert werden, ohne dass dazu der Zugriff auf die Originalsoftware notwendig ist? Werden Composite Applications bei Kunden ausgeliefert und installiert, wird sicherlich die Frage aufkommen, wie sie noch weiter an konkrete Kundenbedürfnisse angepasst werden können. Diese Art der Erweiterung geht dabei über die bisher diskutierten Flexibilitätsansätze hinaus, da Letztgenannte primär auf einer geeigneten Komponentisierung und der dynamischen Konfiguration der Komponenten auch unter Verwendung von Regelwerken beruhte. Komplette Komponenten wurden also ganzheitlich ausgetauscht. Die nicht-invasive, modifikationsfreie Erweiterung um betriebswirtschaftlich benötigte neue Felder hingegen weißt diesbezüglich jedoch eine neue Qualität auf, da sie sich durch sämtliche Schichten einer Verbundanwendung hindurchzieht. Demnach ergeben sich für die Verbundanwendung folgende Konsequenzen:

- Das neue Feld wird in der Composite persistiert: folglich muss entweder die existierende Datenbanktabelle um ein Feld erweitert oder eine neue Datenbanktabelle für die Erweiterungsfelder angelegt und von der Composite verwaltet werden.
- Das neue Feld wird über externe Services befüllt und wieder zurückgeschrieben: folglich ändert sich der Schnittstellenvertrag zwischen Composite und der Servicevertrag-Implementierungsschicht. Dies kann entweder über eine Erweiterung der existierenden Schnittstelle erfolgen oder über einen völlig neuen Service, der sich ausschließlich der neuen Felder annimmt.
- Das neue Feld muss in den Benutzeroberflächen berücksichtigt werden, folglich müssen Composite-Anwender-UIs von ihrer Architektur her auf Erweiterungen vorbereitet sein.
- Das neue Feld muss auf Prozessebene in den Prozesskontext eingehen und an Prozessschritte, die wiederum Services bzw. Benutzeroberflächen kapseln, weitergereicht werden können.
- Das neue Feld hat auch Konsequenzen auf die Geschäftslogik. Diese ist gegebenenfalls zu ergänzen.

Neben diesen unmittelbaren Herausforderungen muss auch der Frage nachgegangen werden, wie zu verfahren ist, wenn der Hersteller der Composite selbst wiederum eine neue Version ausliefert und bei der Software-Aktualisierung auf die vom Kunden vorgenommenen Erweiterungen trifft. Wie sollen also Herstelleränderungen und Kundenerweiterungen zusammengeführt werden?

Der erfolgversprechendste Ansatz zur Beantwortung dieser Fragen scheint in einer konsequenten Anwendung der Objektorientierung zu liegen. Beruht eine Composite durchgängig auf Objekten, so können Kundenerweiterungen durch einfache Ableitung der ursprünglichen Objekte eingebracht werden. Voraussetzung dafür ist die einheitliche Verwendung dieser Objekte auf allen Ebenen einer Verbundanwendung. Neben den Attributen umfassen die Objekte auch die typischen CRUD-Lebenszyklusmethoden (Create, Read, Update, Delete) sowie Suchmethoden. Begleitet wird der Ansatz über ein Framework, über das der Kunde Zugriff auf die veränderbaren Objekte sowie die in der Composite verwendeten Benutzeroberflächen erhält, um entsprechende Änderungen einbringen zu können.

Anhand eines Lieferanten-Objekts soll die Funktionsweise dieser Idee kurz erläutert werden. Das Lieferanten-Objekt umfasst für dieses Beispiel die obligatorischen Felder wie Firmenname, Adresse, Telefonnummer, Faxnummer, eMail usw. Wird das Lieferanten-Objekt lokal in der Composite gespeichert, existiert eine dazugehörige Datenbanktabelle. Wird das Objekt hingegen extern verwaltet, so sorgt eine passende Service-Schnittstelle für das Lesen, Aktualisieren und Löschen des Objekts (externer Servicevertrag). Das Objekt wird über Schnittstellen an die Benutzeroberfläche(n) übergeben und liegt auch im Prozesskontext als Objekt vor. Alternativ könnte im Prozesskontext auch lediglich eine Referenz auf das Originalobjekt gehalten werden, um Speicherplatz zu sparen. Für das im folgenden beschriebene Verfahren spielt das allerdings keine Rolle.

Zur Anpassung der Composite sieht dessen Hersteller nun eine Erweiterung des Lieferanten-Objekts vor, d.h. der Kunde kann jederzeit durch Ableiten des ursprünglichen Objekts neue Felder hinzufügen. In unserem Fall soll beispielsweise die Skype-Kennung hinzugefügt werden, da diese in der ursprünglichen Composite nicht vorgesehen war, sie aber für unseren fiktiven Kunden von hoher Bedeutung ist. Ohne die Entwicklungsumgebung (z.B. Eclipse) selbst verwenden zu müssen, werden dem Kunden durch ein Framework die erweiterbaren Objekte angezeigt. Er kann nun selbst bestimmen, welches Objekt er ergänzen möchte und um welche Felder es sich bei den Erweiterungen handelt. Hat er seine Änderungswünsche eingebracht, sorgt eine Generierungslauf über die gesamte Applikation dafür, dass diese Erweiterung in allen Teilen der Anwendung Berücksichtigung findet:

- Bei lokaler Persistenz: eine ergänzende Datenbanktabelle für die Erweiterungsfelder wird generiert. Die CRUD- und Suchmethoden werden durch das erweiterte Objekt automatische überschrieben, so dass die neuen Methoden bei Aufrufen aus der Composite heraus anstelle der Versionen des Originalobjekts gezogen werden.
- Bei externer Verwaltung: es werden automatisch neue Schnittstellen für die Servicevertrag-Implementierungsschicht generiert, die das neue Feld berücksichtigen.
- Da Objekte auch an die Benutzeroberflächen übergeben werden, stehen dieser die neuen Felder aufgrund des Ableitungsmechanismus ebenfalls automatisch

zur Verfügung. Das Framework erlaubt nun die Ergänzung der Benutzeroberflächen durch separate Laschen für neue Felder.

- Auch der Prozesskontext profitiert von der Objektorientierung. Ob es sich um eine Referenz auf das Lieferanten-Objekt oder um eine komplette Kopie handelt – der Mechanismus funktioniert letztendlich auch hier.
- Neue Geschäftslogik kann dann eingebracht werden, wenn das Framework dedizierte Aussprünge aus der Originalsoftware vorsieht. Dieses Konzept ist in der SAP-Software unter dem Begriff BAdI (Business Add-Ins) bekannt geworden (SAP 2010c) und kann auch hier zum Einsatz kommen. Dabei bleibt die Schnittstelle zum Aussprung stabil und muss in unserem Beispiel erneut das Lieferanten-Objekt als Parameter enthalten. Über derartige Aussprünge kann der Kunde dann neue Logik in die Composite einbringen
- Liefert der Hersteller der Verbundanwendung seinerseits eine neue Version aus, ist über das beschriebene Verfahren ebenfalls für eine konfliktfreie Zusammenführung von Kundenerweiterungen und Herstellersoftware gesorgt. Da aus Sicht des Kunden lediglich ein Objekt abgeleitet wurde, kann diese Ableitung natürlich auch auf ein vom Hersteller modifiziertes Objekt angewandt werden. Eine Neugenerierung wird auch in diesem Fall die Modifikationen in die neue Software einbringen. Da Felder in den Benutzeroberflächen ohnehin in eigenständigen Laschen ausgelagert wurden, kommt es auch hier zu keinem Konflikt. Last but not least wird auch nach wie vor die vom Kunden erstellte Geschäftslogik gezogen, da die Aussprungsschnittstelle per Definition stabil bleibt und ohnehin das Objekt als Übergabeparameter enthält.

Letztendlich schafft die Kombination von...

- Programmierkonventionen für den Hersteller der Verbundanwendung (konsequente Verwendung von Objekten)
- Grundsätzen der Objektorientierung (Ausnutzen der Ableitung von Objekten)
- Bereitstellung eines Frameworks zur externen Erweiterung von Objekten, Benutzeroberflächen und Geschäftslogik sowie eine Neugenerierung der gesamten Applikation aufgrund der vorgenommenen Änderungen

die Basis für einen vielversprechenden Ansatz zur Lösung des Problems der modifikationsfreien Erweiterung von Composite Applications. Allerdings wurde dieser Ansatz in der Form noch nicht implementiert, so dass dessen Funktionsweise nicht verifiziert werden konnte. Von daher bieten sich gerade auf diesem für Kunden so wichtigen Gebiet der Erweiterbarkeit von Verbundanwendungen interessante Ansätze für neue Forschungsprojekte.

Diese Arbeit begann mit einem Zitat von Darwin über die Steigerung der Überlebenswahrscheinlichkeit in einem von Veränderungen geprägten Umfeld durch schnellstmögliche Anpassung. Verbundanwendungen sind eine Antwort auf diese Herausforderung, um Unternehmen eine umgehende Umsetzung der von der Firmenleitung vorgegebenen Strategie zu ermöglichen und somit für ihr Überleben zu sorgen. Dabei unterstützt die beschriebene Herangehensweise zum Entwurf und zur Implementierung von Verbundanwendungen eine nachhaltige Architektur, die auch nach mehreren Jahren der Weiterentwicklung wartbar bleibt. Der Grundstein ist also gelegt. Auf dieser Basis können nun weitere Forschungsarbeiten zur Perfektionierung sowohl der Architektur als auch der Methodik vorangetrieben werden.

# 7 Abbildungsverzeichnis

## 8 Tabellenverzeichnis

# Tabellenverzeichnis

# 9 Abkürzungsverzeichnis

| Abkürzung | Erläuterung |
|---|---|
| 2PC | Two-Phase-Commit |
| ABAP | Advanced Business Application Programming |
| ADEPT | Application Development based on Encapsulated pre-modeled Process Templates |
| BAPI | Business Application Programming Interface |
| BAL | Backend Abstraction Layer |
| BPD | Business Process Diagram |
| BPEL | Business Process Execution Language |
| BPM | Business Process Management |
| BPMI | Business Process Management Initiative |
| BPMN | Business Process Model and Notation |
| BPMS | Business Process Management System |
| BRG | Business Rules Group |
| BRM | Business Rules Management |
| BW | Business Warehouse |
| CAF | Composite Application Framework |
| CCTS | Core Component Technical Specification |
| CRM | Customer Relationship Management |
| CRUD | Create, Read, Update, Delete |
| CSV | Comma Separated Values |
| EAI | Enterprise Application Integration |
| EIP | Enterprise Integration Pattern |
| EJB | Enterprise Java Beans |
| EPK | Ereignisgesteuerte Prozessketten |
| ERP | Enterprise Resource Planning |
| ESB | Enterprise Service Bus |
| ESR | Enterprise Services Repository |
| FEH | Forward Error Handling |
| FPN | Federated Portal Network |
| GDT | Global Data Types |
| GoF | Gang-of-Four |
| HTTP | HyperText Transfer Protocol |
| iCOD | Industry Composite Development |
| ISV | Independent Software Vendor |
| Java EE | Java Platform Enterprise Edition |
| Java SE | Java Platform Standard Edition |
| JAX-WS | Java API for XML Web Services |

| Abkürzung | Erläuterung |
|---|---|
| JMX | Java Management Extensions |
| JPA | Java Persistence API |
| JSF | Java Server Faces |
| JSP | JavaServer Pages |
| KMC | Knowledge Management and Collaboration |
| KMU | Kleine und mittelständische Unternehmen |
| LDAP | Lightweight Directory Access Protocol |
| MDA | Model Driven Architecture |
| MDD | Model Driven Development |
| MDM | Master Data Management |
| MEP | Message Exchange Pattern |
| MOM | Message Oriented Middleware |
| NWDI | SAP NetWeaver Development Infrastructure |
| NWDS | SAP NetWeaver Developer Studio |
| OAGi | Open Applications Group |
| OAGIS | Open Applications Group Integration Specification |
| P2P | Point-to-Point |
| POC | Proof of Concept |
| POJO | Plain Old Java Object |
| QoS | Quality of Service |
| RCA | Rich Client Application |
| REST | REpresentational State Transfer |
| RFC | Remote Function Call |
| RFM | Remote Function Module |
| RPC | Remote Procedure Call |
| SaaS | Software-as-a-service |
| SAP NetWeaver BPM | SAP NetWeaver Business Process Management |
| SAP NetWeaver BRM | SAP NetWeaver Business Rules Management |
| SAP NetWeaver CE | SAP NetWeaver Composition Environment |
| SAP NetWeaver VC | SAP NetWeaver Visual Composer |
| SDO | Service Data Objects |
| SHIL | Service-Human-Interaction-Layer |
| SLA | Service Level Agreement |
| SOA | Service Oriented Architecture |
| SOAP | Simple Object Access Protocol |
| SQL | Structured Query Language |
| S.W.I.F.T. | Society for Worldwide Interbank Financial Telecommunication |
| SWOT | Strength, Weaknesses, Opportunities, Threads |
| SysML | Systems Modeling Language |

| Abkürzung | Erläuterung |
|---|---|
| TCO | Total cost of ownership |
| UDDI | Universal Description, Discovery, and Integration |
| UI | User Interface |
| UN/CEFACT | United Nations Centre for Trade Facilitation and Electronic Business |
| URL | Uniform Resource Locator |
| UTA | User Trained Agents |
| VC | Visual Composer |
| VCD | Variant Component Diagram |
| WSDL | Web Services Description Language |
| W3C | World Wide Web Consortium |
| WTP | Web Tools Platform |
| WYSIWYG | What You See Is What You Get |
| XML | Extensible Markup Language |
| XSD | XML Schema Definition |
| XSLT | Extensible Stylesheet Language Transformations |

# 10 Literaturverzeichnis


ADEPT2 (2010): Web-Seite der Universität Ulm zum ADEPT2-Forschungsprojekt. Letzter Zugriff am 21.09.2010 unter http://www.uni-ulm.de/in/iui-dbis/forschung/projekte/adept2.html

Aigner, W (2010): *Make Timeless Software Principles Tangible*. Präsentation gehalten anlässlich des Developer Kick-Off Meetings (DKOM) der SAP. (11.03.2010)

Allweyer, T. (2009a): *BPMN 2.0 Business Process Model and Notation. Einführung in den Standard für die Geschäftsprozessmodellierung*. Norderstedt: Books on Demand

Allweyer, T. (2009b): *Kollaborationen, Choreographien und Konversationen in BPMN 2.0: Erweiterte Konzepte zur Modellierung übergreifender Geschäftsprozesse*. Letzter Zugriff am 31.08.2010 unter http://kurze-prozesse.de/blog/wp-content/uploads/2009/06/kollaborationen-choreographien-und-konversationen-in-bpmn-20.pdf

Allweyer, T. (2010a): *Unternehmen als Prozess Engine? Möglichkeiten und Grenzen mit BPMN*. Letzter Zugriff am 01.11.2010 unter http://www.slideshare.net/bpmn2010/bpmn2010-allweyer-5557956?from=ss_embed

Allweyer, T. (2010b): *Adaptive Case Management – Informativer Sammelband*. Letzter Zugriff am 12.11.2010 unter http://www.kurze-prozesse.de/2010/07/21/adaptive-case-management-informativer-sammelband/#more-462

Anstey, J. (2009): *Apache Camel: Integration Nirvana*. Letzter Zugriff am 19.05.2011 unter http://architects.dzone.com/articles/apache-camel-integration

Balko, S. (2009): *Workflow Pattern Coverage in SAP NetWeaver BPM 7.11*. Letzter Zugriff am 12.10.2010 unter http://www.sdn.sap.com/irj/scn/go/portal/prtroot/docs/library/uuid/8026f177-f32f-2c10-b7b0-9cc31d92984d?QuickLink=index&overridelayout=true

Balko, S. (2010a): *Understanding Intermediate Events, Asynchronous Message Receipt and Correlation in NetWeaver BPM 7.20*. Letzter Zugriff am 15.09.2010 unter http://www.sdn.sap.com/irj/scn/index?rid=/library/uuid/f0369539-e876-2d10-ef93-8f30e112ee6d&overridelayout=true

Balko, S. (2010b): *Enjoy NetWeaver BPM - Part 3: Workflow Patterns Reloaded - NetWeaver BPM 7.20.* Letzter Zugriff am 12.10.2010 unter http://www.sdn.sap.-com/irj/scn/weblogs?blog=/pub/wlg/17575

Banerjee, A. (2006): *What are Composite Applications?* Letzter Zugriff am 12.07.2010 unter http://msdn.microsoft.com/en-us/architecture/bb220803(printer).aspx

Baeyens, T. (2010): *Alive and Kicking*. Letzter Zugriff am 21.04.2011 unter http://process-developments.blogspot.com/2010/03/alive-and-kicking.html.

Bloomberg, J., Schmelzer, R (2006): *Service Orient or Be Doomed!: How Service Orientation Will Change Your Business*. John Wiley & Sons

BRG - Business Rules Group (2003): *The business rules manifesto. The principals of rules independence.* Letzter Zugriff am 23.10.2010 unter http://www.businessrules-group.org/brmanifesto.htm

Burlton, R. T. (2001): *Business Process Management: Profiting from Success*. Indianapolis: Sams Publishing.

Camel (2011): *Apache Camel Homepage*. Letzter Zugriff am 16.05.2011 unter http://camel.apache.org/

Dadam, P., Reichert, M., Rinderle-Ma, S., Göser, K., Kreher, U., Jurisch, M. (2009): *Von ADEPT zur AristaFlow® BPM Suite – Eine Vision wird Realität. "Correctness by Construction" und flexible, robuste Ausführung von Unternehmensprozessen*. Technical Report UIB-2009-02, University of Ulm, Faculty of Electrical Engineering and Computer Science.

EABPM (2009): *Business Process Management Common Body of Knowledge - BPM CBOK: Leitfaden für das Prozessmanagement herausgegeben von der EABPM (European Association of Business Process Management).* Schmidt Dr. Goetz, Wettenberg, 2009

EIP (2011): *Enterprise Integration Patterns Tools and Downloads*. Letzter Zugriff am 16.05.2011 unter http://enterpriseintegrationpatterns.com/downloads.html

Eller, B. (2009): *Systematische Integration des Usability Engineering in die Anwendungssystementwicklung zwecks Unterstützung einer nutzerorientierten Entwicklungsarbeit*. Darmstadt: Dissertationsschrift Universität Darmstadt

Erl, T. (2008): *SOA Design Patterns*. Prentice Hall

Erl, T., Arsanjani, A., Booch, G., Boubez, P., Chappell, D., deVadoss, J., Josuttis, N., Krafzig, D., Little, M., Loesgen, B., Thomas Manes, A., McKendrick, J., Ross-Talbot, S., Tilkov, S., Utschig-Utschig, C., Wilhelmsen, H. (2009): *SOA Manifesto*. Letzter Zugriff am 19.10.2010 unter http://soa-manifesto.org/

Fildebrandt, U., Schubert, H., Hoursanov, A., Drabant, B., Vatkov, B., Simeonov, E., Dick, E., Ittel, J., Vasudevan, K., Steiner, M., Kabadzhov, N., Tatarova, P., Keil, U., Stiehl, V., Reddy, P. K. (2010): *Composite Development Architecture Guidelines*. Letzter Zugriff am 17.04.2011 unter

http://www.sdn.sap.com/irj/scn/go/portal/prtroot/docs/library/uuid/109b805f-2e28-2d10-ed9c-94eea0e8ae5c?QuickLink=index&overridelayout=true

Fowler, M. (2002): *Patterns of Enterprise Application Architecture*. Amsterdam: Addison Wesley

Fischer, M., Link, M., Ortner, E., Zeise, N. (2010): *Servicebase Management Systems: A Three-Schema-Architecture for Service-Management*. In (Fähnrich, K.-P. Hrsg.): Informatik 2010 - Service Science. Neue Perspektiven für die Informatik, 27.09 - 01.10. 2010, Leipzig. Ges. für Informatik, Bonn, 2010; S. 730–735.

Freund, J., Rücker, B., Henninger, T. (2010): *Praxishandbuch BPMN*. München: Hanser Verlag

Frotscher, T. (2009): *Trugschluss*. Java Magazin 6/2009, S. 99-104. Frankfurt am Main: Software & Support Verlag.

FuseSource (2011a): *Enterprise Integration Pattern reference*. Letzter Zugriff am 16.05.2011 unter http://fusesource.com/docs/ide/camel/1.0/eip_ref/index.html

FuseSource (2011b): *Enrich Pattern*. Letzter Zugriff am 16.05.2011 unter http://fusesource.com/docs/ide/camel/1.0/eip_ref/enrich.html

Gaitanides, M. (1983): *Prozessorganisation. Entwicklung, Ansätze und Programme prozessorientierter Organisationsgestaltung*. Vahlen, München 1983

Gamma, E., Helm, R., Johnson, R., Vlissides, J. (1995): *Design Patterns: Elements of Reusable Object-Oriented Software*. Massachusetts: Addison-Wesley Reading

Golega, S. T. (2007): *Composition and Coordination of Transactional Business Processes*. Potsdam: Master Thesis Hasso-Plattner-Institut Universität Potsdam

Graml, T., Bracht, R., Spies, M. (2008): *Patterns of Business Rules to Enable Agile Business Processes*. Enterprise Information Systems, 2(4):385{402, November 2008.

Grimm, H., Speck, A.., Stiehl, V. (2009): *SOA Experience Workshop (1st version). Module 1: Introduction to Service Oriented Architecture.* Präsentation zur Einführung von Composite Applications in Unternehmen.

Grimm, H., Speck, A.., Stiehl, V. (2010): *SOA Experience Workshop (2nd version). Module 4: Loosely Coupled Process Coordination*. Präsentation zur Einführung von Composite Applications in Unternehmen.

Grobe, A. (2009): *Checklist to identify valuable use cases*. Präsentation gehalten auf der SAP World Tour 2009 (08.06.2009)

Grollius, T. (2010): *Interaktive Konfigurierung dynamischer Anwendungssysteme aus Komponenten*. In (Engels, G.; Luckey, M.; Schäfer, W. Hrsg.): Proceedings - Soft-

ware Engineering 2010 - Workshopband ( inkl. Doktorandensymposium ). Zur Tagung 22.-26. Februar 2010, Paderborn, Bonn, 2010; S. 75–85.

Hammer, M., Champy, J. (1993): *Reengineering the Corporation*. Harper Business 1993

Hill, M., Dimitrova, D. (2008): *Exceptional Scheduling of Shift Workers*. Letzter Zugriff am 14.07.2010 unter http://www.sdn.sap.com/irj/scn/index?rid=/library/uuid/40ab3554-58b1-2b10-ceb9-80861dacd5cf&overridelayout=true

Hofmann, F. (1984): *Betriebssysteme: Grundkonzepte und Modellvorstellungen*. Stuttgart: B. G. Teubner

Hohpe, G., Woolf, B. (2004): *Enterprise Integration Patterns. Designing, Building, and Deploying Messaging Solutions.* Boston: Addison Wesly

Hümmer, W. (2004): *Vertragsverhandlungen um konfigurierbare Produkte im elektronischen Handel.* Dissertation. Universität Erlangen-Nürnberg, Technische Fakultät, 2004.

Ibsen, C., Anstey, J. (2011): *Camel in Action*. Stamford: Manning Publications

Jablonski, S., Bussler, C. (1996): *Workflow Management Systems: Modelling Concepts, Architecture and Implementation*. Thomson Learning, 1996.

Jablonski, S. (2008): *Prozessdesign und -modellierung für ein holistisches Prozessmanagement*. In Elisabeth Heinemann und Erich Ortner, Hrsg., Anwendungsinformatik: Die Zukunft der Enterprise Engineering ; Festschrift für Erich Ortner zum 60. Geburtstag, Seiten 83–104. Nomos-Verl.-Ges., Baden-Baden, 2008.

Josuttis, N. (2008): *SOA in der Praxis*. Heidelberg: dpunkt.

Josuttis, N. (2010): *Das SOA-Manifest – Kontext, Inhalt, Erläuterung*. Heidelberg: dpunkt.

Krafzig, D., Banke, K. & Slama, D. (2004): *Enterprise SOA – Service-Oriented Architecture Best Practices*. Prentice Hall

Lautenbacher, F. (2009): *Semantic Business Process Modeling: Principles, Design Support and Realization*. Augsburg: Dissertationsschrift Universität Augsburg

Lehmann, F. R. (1999): *Fachlicher Entwurf von Workflow-Management-Anwendungen*. Teubner, Stuttgart, 1999.

Link, M. (2010): *Zweistufiger Modellierungsansatz zum nachhaltigen Prozessmanagement.* In (Ortner, E. Hrsg.): Konstruktive Informatik. Modellierung und Entwicklung von Anwendungssystemen, 2010; S. 33–42.

Malek, T., Dimitrova, D. (2008): *Master Data Management: Create Customer Data*. Letzter Zugriff am 14.07.2010 unter http://www.sdn.sap.com/irj/scn/go/portal/prtroot/docs/library/uuid/005808cd-59b1-2b10-b589-c7016d221092?QuickLink=index&overridelayout=true

Maier, B., Normann, H., Trops, B., Utschig-Utschig, C., Winterberg, T. (2009a): *Das kanonische Datenmodell*. Java Magazin 9/2009, S. 98-112. Frankfurt am Main: Software & Support Verlag

Maier, B., Normann, H., Trops, B., Utschig-Utschig, C., Winterberg, T. (2009b): *Lose Kopplung – Warum das Loslassen verbindet*. Java Magazin 3/2009, S. 84-91. Frankfurt am Main: Software & Support Verlag

Maier, B., Normann, H., Trops, B., Utschig-Utschig, C., Winterberg, T. (2009c): *SOA und Benutzeroberflächen*. Java Magazin 8/2009, S. 88-98. Frankfurt am Main: Software & Support Verlag

Manes, A. T. (2009): *SOA is Dead; Long Live Services*. Letzter Zugriff am 05.07.2010 unter http://apsblog.burtongroup.com/2009/01/soa-is-dead-long-live-services.html.

Milanovic M., Gasevic, D., Wagner, G.: *Combining Rules and Activities for Modeling Service-Based Business Processes.* In EDOC, Munich, Germany, 2008.

Miller, J., Mukerji, J. (2003): *MDA guide*. Technical Report omg/2003-06-01, Object Management Group (OMG), June 2003. Version 1.0.1

OAGIS (2009): *OAGIS Release 9.4.1*. Letzter Zugriff am 29.07.2010 unter http://www.oagi.org/dnn2/DownloadsandResources/OAGIS941.aspx

OMG (2010a): *Business Process Model And Notation, V2.0 Beta 2*. Letzter Zugriff am 21.07.2010 unter http://www.omg.org/cgi-bin/doc?dtc/10-06-04.
Alternativ: http://www.omgwiki.org/bpmn2.0-ftf/lib/exe/fetch.php?id=public%3Ahome&cache=cache&media=public:bpmn_2-0_dtc-2010-05-03_-_no_markup.pdf

OMG (2010b): *BPMN 2.0 by Example*. Letzter Zugriff am 25.08.2010 unter http://www.omg.org/cgi-bin/doc?dtc/10-06-02.pdf

OMG (2011): *Business Process Model And Notation (BPMN), V2.0*. Letzter Zugriff am 20.06.2011 unter http://www.omg.org/spec/BPMN/2.0/PDF/

Ortner, E (2005a): *Component-based Application Architecture for Enterprise Information Systems*. In (Härder, T., Lehner, W. Eds.): Data Management in a Connected World. Springer, Heidelberg, 2005; S. 181-200.

Peter, M. (2009): *Taking SOA to the Next Level – Asynchronous Enterprise Services*. Präsentation gehalten anlässlich der SAP TechEd 2009. (27.-29.10.2009)

Pitschke, J. (2010): *Unternehmensmodellierung für die Praxis. Eine Einführung in die Darstellung von Unternehmensmodellen.* Norderstedt: Books on Demand

Pucher, M.J. (2010): *Adaptive Prozesse: Theorie und Praxis.* Letzter Zugriff am 12.11.2010 unter http://adaptiveprocess.wordpress.com/2010/05/31/adaptive-prozesse-theorie-und-praxis/

Rauscher, J. & Stiehl, V. (2008): *The Developer's Guide to SAP NetWeaver Composition Environment*. Bonn: Galileo Press.

Ross, R. G. (2009): *Business Rule Concepts 3rd Edition. Getting to the Point of Knowledge.* Business Rule Solutions, LLC

Rupp, C. (2009): *Requirements-Engineering und -Management: Professionelle, iterative Anforderungsanalyse für die Praxis.* Hanser Fachbuch, 2009

SAP (2002a): *SAP Launches Cross Applications - New Breed of Cross-Functional Business Applications Drives Continuous Business Improvement*. Letzter Zugriff am 01.03.2011 unter http://www.sap.com/about/newsroom/press.epx?pressID=1334

SAP (2002b): *SAP Delivers mySAP™ Technology and Details SAP xApps™ Vision*. Letzter Zugriff am 01.03.2011 unter http://www.sap.com/about/newsroom/press.epx?pressID=1634

SAP (2002c): *SAP xApps™ Gain Market Momentum - First Customer Deploys xApp to Deliver Increased Visibility to Plant Assets; SAP Announces New SAP xApp to Support Product Design and Development*. Letzter Zugriff am 21.04.2011 unter http://www.sap.com/press.epx?pressID=1636

SAP (2005): *Architecture Guideline for Model-Driven Composite Development in the Composition Environment V1.0*. SAP internes Papier zur Entwicklung von Composite Applications (Veröffentlicht in Schuller 2007a, 2007b, 2007c).

SAP (2007): *Architecture Guideline for Model-Driven Composite Development in the Composition Environment V2.2*. SAP internes Papier zur Entwicklung von Composite Applications (Veröffentlicht in Schuller 2007a, 2007b, 2007c).

SAP (2008): *Architecture Guideline for Model-Driven Composite Development in the Composition Environment V2.3*. SAP internes Papier zur Entwicklung von Composite Applications.

SAP (2009a): *SAP Global Data Type Design Methodology*. Letzter Zugriff am 29.07.2010 unter http://wiki.sdn.sap.com/wiki/display/GDT/SAP+Global+Data+Type+Design+Methodology

SAP (2009b): *SAP Global Data Type Catalog*. Letzter Zugriff am 29.07.2010 unter http://wiki.sdn.sap.com/wiki/display/GDT/SAP+Global+Data+Type+Catalog

SAP (2009c): *SAP Guidelines for Best-Built Applications that Integrate with SAP Business Suite*. Letzter Zugriff am 01.03.2011 unter http://www.sdn.sap.com/irj/sdn/best-builtapps

SAP (2010a): *BPM Use Case – Handling issue reports submitted by citizens*. Letzter Zugriff am 19.07.2010 unter http://wiki.sdn.sap.com/wiki/display/BPX/BPM+Use+Cases+-+Handling+issue+reports+submitted+by+citizens

SAP (2010b): *Enterprise Services Repository*. Letzter Zugriff am 23.04.2011 unter http://www.sdn.sap.com/irj/sdn/nw-esr.

SAP (2010c): *Business Add-Ins (BAdIs)*. Letzter Zugriff am 20.11.2010 unter http://help.sap.com/saphelp_nwpi71/helpdata/de/8f/f2e540f8648431e10000000a1550b0/frameset.htm

SAP (2010d): *Forward Error Handling.* Letzter Zugriff am 23.04.2011 unter http://eswork-place.sap.com/socoview(bD1lbiZjPTAwMSZkPW1pbg==)/render.asp?sap-unique=044301&sap-params=aWQ9NjIxMTAwQkFGRDg0NDNCMEI2OTI3QURG-M0Y2RDcyRkEmcGFja2FnZWlkPURFMDQyNkREOUIwMjQ5RjE5NTE1MDAxQ-TY0RDNGNDYy

SAP (2010e): *Forward Error Handling – Dokumentation.* Letzter Zugriff am 23.04.2011 unter http://help.sap.com/saphelp_nwes72/helpdata/DE/cd/798aa3c7754c61b2f2d50ea7b66aac/content.htm

SAP (2011): *SAP Streamwork.* Letzter Zugriff am 23.04.2011 unter http://www.sapstream-work.com/

Schacher, M., Grässle, P. (2006): *Agile Unternehmen durch Business Rules*. Berlin: Springer Verlag

Schuller, A. (2007a): *Architecture Guideline Series for Composite Applications. Introduction and Basic Overview*. Letzter Zugriff am 17.04.2011 unter http://www.sdn.sap.-com/irj/scn/go/portal/prtroot/docs/library/uuid/1078e3b0-ec5d-2a10-f08a-c9b878917b19?QuickLink=index&overridelayout=true

Schuller, A. (2007b): *Architecture Guideline Series for Composite Applications. Portal and Process Layer.* Letzter Zugriff am 17.04.2011 unter http://www.sdn.sap.com/irj/scn/go/portal/prtroot/docs/library/uuid/1078e3b0-ec5d-2a10-f08a-c9b878917b19?QuickLink=index&overridelayout=true

Schuller, A, (2007c): *Architecture Guideline Series for Composite Applications. Business Logic, Abstraction Layer and Connectivity*. Letzter Zugriff am 17.04.2011 unter http://www.sdn.sap.com/irj/scn/go/portal/prtroot/docs/library/uuid/00caf8bd-487a-2a10-36a9-93d840309310?QuickLink=index&overridelayout=true

Silver, B. (2008): *Three Levels of Process Modeling with BPMN*. Letzter Zugriff am 22.07.2010 unter http://www.brsilver.com/three-levels-of-process-modeling-with-bpmn/

Silver, B. (2009): *BPMN Method & Style*. Aptos CA: Cody-Cassidy Press

Stein, S. (2009): *Modelling Method Extension for Service-Oriented Business Process Management*. Kiel: Dissertationsschrift Christian-Albrechts-Universität Kiel

Stein, S. (2010): *EPC vs. BPMN – the perfect flamewar*. Letzter Zugriff am 01.09.2010 unter http://www.ariscommunity.com/users/sstein/2010-04-15-epc-vs-bpmn-perfect-flamewar

Stiehl, V. (2006): *Guidelines for Specifying Composite Applications*. Letzter Zugriff am 20.07.2010 unter https://www.sdn.sap.com/irj/scn/index?rid=/library/uuid/20844e88-0d01-0010-de9a-eb2d302df7b7&overridelayout=true

Stiehl, V., Deng, J. (2008): *Project Issue Management*. Letzter Zugriff am 14.07.2010 unter http://www.sdn.sap.com/irj/scn/go/portal/prtroot/docs/library/uuid/0087457d-65b1-2b10-c1b0-93f02bd60434?QuickLink=index&overridelayout=true

Stiehl, V. (2009a, 2009b, 2009c, 2009d): *BPMN - Vermittler zwischen den Welten.*
*Teil 1: Was ist BPMN?* Letzter Zugriff am 21.07.2010 unter http://it-republik.de/jaxenter/artikel/BPMN---Vermittler-zwischen-den-Welten-Teil-1-2531.html
*Teil 2: Prozessflussmodellierung, Zuweisung von Aktivitäten und Rollen.* Letzter Zugriff am 21.07.2010 unter http://it-republik.de/jaxenter/artikel/BPMN---Vermittler-zwischen-den-Welten---Teil-2-2541.html
*Teil 3: Datenflussmodellierung, Gateway-Definition und Fehlerbehandlung.* Letzter Zugriff am 21.07.2010 unter http://it-republik.de/jaxenter/artikel/BPMN---Vermittler-zwischen-den-Welten-Teil-3-2536.html
*Teil 4: Ausführung des BPMN-Prozesses* Letzter Zugriff am 21.07.2010 unter http://it-republik.de/jaxenter/artikel/BPMN---Vermittler-zwischen-den-Welten---Teil-4-2547.html

Stiehl, V. (2009e): *SAP Guideline for Best-Built Applications: One short sentence with huge implications*. Letzter Zugriff am 01.08.2010 unter http://www.sdn.sap.com/irj/scn/weblogs?blog=/pub/wlg/16224

Stiehl, V. (2010a): *Take a Serious Look at the "A" in SOA and Gain Flexible, Adaptable Architecture*. SAP Professional Journal - Volume 12 (2010), Update 2. Dedham, MA: Wellesly Information Services (WIS)

Stiehl, V. (2010b): *Lose Kopplung und BPMN – Passt das?* Java Magazin 2/2010, S. 67-73. Frankfurt am Main: Software & Support Verlag

Stiehl, V. (2010c): *SAP NetWeaver Composition Environment 7.2: New Feature Improves Development of Loosely Coupled SOA-Based Applications*. SAP Professional Journal - Volume 12 (2010), Update 2. Dedham, MA: Wellesly Information Services (WIS)

Swenson, K. D. (2010): *Mastering the Unpredictable: How Adaptive Case Management Will Revolutionize the Way That Knowledge Workers Get Things Done.* Meghan-Kiffer Press.

SysML (2010): *OMG Systems Modeling Language, Version 1.2, OMG Document Number: formal/2010-06-01*. Object Management Group, 2010

UN/CEFACT (2003): *Core Components Technical Specification – Part 8 of the ebXML Framework. Version 2.01*. Letzter Zugriff am 29.07.2010 unter http://www.unece.org/cefact/ebxml/CCTS_V2-01_Final.pdf

van der Aalst, W., ter Hofstede, A., Kiepuszewski, B., Barros, A. (2003): *Workflow Patterns.* Distributed and Parallel Databases, 14(3):5-51

van der Aalst (2010): *Process Mining – Reasearch, Tools, Applications.* Letzter Zugriff am 12.11.2010 unter http://www.processmining.org

Vanderhaeghen, D., Fettke, P., Loos, P. (2010): *Organisations- und Technologieoptionen des Geschäftsprozessmanagements aus der Perspektive des Web 2.0.* Wirtschaftsinformatik 1|2010; S. 17-31

W3C (2004): *XML Schema Part 2: Datatypes Second Edition*. Letzter Zugriff am 09.09.2010 unter http://www.w3.org/TR/xmlschema-2/

Waldo, J., Wyant, G., Wollrath, A., Kendall, S. (1994): *A Note on Distributed Computing*. Technical Report, SMLI TR-94-29. Sun Microsystems Laboritories, November 1994. Letzter Zugriff am 06.05.2011 unter http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.41.7628

Wedekind, H., Müller, T. (1981): *Stücklistenorganisation bei einer großen Variantenzahl.* Angewandte Informatik, 23 (9) 1981; S. 377-383

Wedekind, H. (1989): *Konstruktionserklären und Konstruktionsverstehen. Elemente des Aufbaus eines Konstruktionsführungssystems.* Zeitschrift für wirtschaftliche Fertigung und Automatisierung, 84 (1) 1989; S. 623-629

White, S. (2004): *Process Modeling Notations and Workflow Patterns.* Letzter Zugriff am 12.10.2010 unter http://www.bpmn.org/Documents/Notations_and_Workflow_Patterns.pdf

Winter, R. (2009): *Patterns in der Wirtschaftsinformatik*. Wirtschaftsinformatik 6/2009, S. 535-542.

Woods, D. (2003): *Packaged Composite Applications*. Sebastopol: O'Reilly

Woods, D. & Mattern T. (2006): *Enterprise SOA: Designing IT for Business Innovation*, Sebastopol: O'Reilly

Zur Mühlen, M. (2004): *Organizational Management in Workflow Applications – Issues and Perspectives*. Information Technology and Management 5, 271–291, 2004

# Erklärung

Hiermit erkläre ich, dass ich die vorliegende Arbeit selbstständig und ohne Benutzung anderer als der angegebenen Hilfsmittel angefertigt habe. Alle Stellen, die wörtlich oder sinngemäß aus veröffentlichten und nicht veröffentlichten Schriften entnommen wurden, sind als solche kenntlich gemacht. Die Arbeit ist in gleicher oder ähnlicher Form oder auszugsweise im Rahmen einer anderen Prüfung noch nicht vorgelegt worden.

Möhrendorf, den 07. Juli 2011

_____

Volker Stiehl